AD-A090 606 MITRE CORP BEDFORD MA F/G 17/2.1
THE MULTIPLE STACK ALGORITHM IMPLEMENTED ON A ZILOG Z-80 MICROC--ETC(U)
JUL 80 H H MA F19628-79-C-0001
UNCLASSIFIED MTR-3843 RADC-TR-80-232 NL

1 OF 1
AD
A090606

END
DATE
FILMED
11-80
DTIC

LEVEL II

(12)

RADC-TR-80-232
Interim Report
July 1980

AD A090606

THE MULTIPLE STACK ALGORITHM IMPLEMENTED ON A ZILOG Z-80 MICROCOMPUTER

The MITRE Corporation

Howard H. Ma

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-80-232 has been reviewed and is approved for publication.

APPROVED: Jerry Silverman

JERRY SILVERMAN
Project Engineer

APPROVED: Clarence D. Turner

CLARENCE D. TURNER, Acting Director
Solid State Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

[illegible]

[illegible]

[illegible]

[illegible]

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(12) 1. REPORT NUMBER: RADC-TR-80-232

2. GOVT ACCESSION NO.: AD-A090 606

3. RECIPIENT'S CATALOG NUMBER:

(6) 4. TITLE (and Subtitle): THE MULTIPLE STACK ALGORITHM IMPLEMENTED ON A ZILOG Z-80 MICROCOMPUTER,

(9) 5. TYPE OF REPORT & PERIOD COVERED: Interim Report,

(14) 6. PERFORMING ORG. REPORT NUMBER: MTR-3843

(10) 7. AUTHOR(s): Howard H./Ma

(15) 8. CONTRACT OR GRANT NUMBER(s): F19628-79-C-0001

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
The MITRE Corporation
Bedford MA 01730

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
61102F
(16) 2305J128 (17) J1

11. CONTROLLING OFFICE NAME AND ADDRESS:
Deputy for Electronic Technology (RADC/ESE)
Hanscom AFB MA 01731 (11)

12. REPORT DATE: Jul[redacted]80 (12) 95

13. NUMBER OF PAGES: 97

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Jerry Silverman (RADC/ESE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Multiple stack algorithm
convolutional codes
decoding
Zilog (Z-80) microcomputer implementation.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A new sequential decoding algorithm, the multiple stack algorithm or MSA, has been implemented and tested on a Zilog Z-80 microcomputer system. The results of this implementation, when compared to those of a (2,1) 8 Viterbi algorithm implemented on the same microcomputer system, show that the MSA obtains significantly higher data throughput. These results suggest the feasibility of a real-time implementation of codes with along constraint lengths (in excess of eight) on small, low-cost microprocessors.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

235050

Acknowledgements

The author gratefully acknowledges S. S. Weinrich for his expert help in completing the memory enhancement task, and R. Drury for his excellent programming and hardware designing assistance throughout the course of this work.

He would also like to thank D. O. Carhoun, R. J. Cosentino, R. D. Haggarty, T. J. McDonald, E. A. Palo, and E. N. Skoog, whose support and encouragement have been instrumental in bringing this work to a satisfactory conclusion.

Accession For	
NTIS GRA&I	X
DDC TAB	
Un[illegible]	
[illegible]	
[illegible]	
B[illegible]	
[illegible]	
[illegible] Codes	
Dist.	Avail and/or special
A	

PREFACE

This report presents the essential results of our experimentation in implementing a new convolutional decoding algorithm on an inexpensive Zilog Z-80 microcomputer system. The work reported was performed in FY 79 as part of the MITRE [redacted] project 7010: Low Cost Electronics.

The multiple stack algorithm (MSA) was designed by Chevillat and Costello and first reported in 1977 [1]. Their work is the basis from which we began our studies. The experiment gave us the opportunity to carefully examine and evaluate the performance of the Z-80 in a complicated simulation. The implications of the microcomputer's performance apply directly to a major, on-going consideration of the Low Cost Electronics project: the effective utilization of inexpensive LSI microprocessors in signal processing tasks.

The report will deal with a specialized topic and certain terms used frequently throughout the text may not be familiar to the non-specialist. Interested readers are referred to the reference works of Wozencraft and Jacobs [19], Gallager [20], Peterson and Weldon [21], and Viterbi and Omura [22], to supplement our explanations where necessary.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Concluded)

LIST OF TABLES

SECTION I

INTRODUCTION

1.1 Purpose

A continuing concern of the Low Cost Electronics project is the effective use of commercially available large scale integrated microprocessors and associated memory in suitable signal processing tasks. Because of our interest in the practical application of error correction coding, and the software-intensive aspects of sequential decoding of convolutional codes, we selected a sequential decoding algorithm as an application example. Among the various decoding algorithms available, we have focused our attention on a recently developed sequential decoding algorithm, the Multiple Stack Algorithm (MSA) [1], because of its expected performance and since its decoding operations seemed quite suitable for microprocessor implementation.

This report examines in some detail the MSA and its implementation on a low-cost, general-purpose small computer, typified by the Zilog Z-80 microprocessor.

The MSA is an improved form of sequential decoding which reduces erasure probabilities and obtains potentially low undetected error probabilities with a modest decoding effort. MSA decoding complexity tends to be independent of the code constraint length which means that, when sufficient memory is available, high performance codes with larger encoder constraint lengths can be used. MSA, which achieves nearly as equal an error rate as maximum-likelihood decoding at faster throughput for similar levels of decoder complexity, may be one of the most effective alternatives to the popular Viterbi decoding algorithm for convolutional codes.

1.2 Background

The Low-Cost Electronics work in digital error control recognizes that error-correction coding, used in conjunction with spread-spectrum

modulation, provides an added dimension to the design of jam-resistant communications systems. Our work addresses the perplexing dilemma of error-correction coding performance (or coding gain) versus cost and complexity of implementation. Our work concerns the effective use and implementation of both convolutional and linear algebraic block codes. This report addresses our work in convolutional decoding.

Convolutional codes are frequently described as tree codes because the set of possible code words can often be pictured as a tree-like structure, each code word being represented by a distinct branch emanating from the root node. Probabilistic decoding algorithms have been devised that attempt to select a branch in the tree minimally distant from the received sequence of code symbols that has been corrupted by noise. These algorithms separate into two classes: maximum likelihood decoding which makes an optimal decision at each branch point in an estimated code tree, and sequential decoding which attempts to select an optimum path through the tree by successive iteration that frequently requires back-search.

The maximum likelihood algorithms, represented most typically by the well-known Viterbi algorithm, require all the nodes in the tree to be examined. Consequently, they are limited to relatively short codes by the available processor storage capacity (and access time), this storage requirement expanding exponentially with code constraint length.

The sequential algorithms search the code tree structure for correct codewords on a trial-and-error basis. Incorrect paths are identified and rejected.

Both sequential and Viterbi decoding offer practical choices to a communication engineer designing a high performance, efficient,

communication system, but the usefulness of both methods has been limited by the decoding complexity and speed. For Viterbi decoding, both the computational complexity and decoding effort are proportional to e^{ν} (where ν is the constraint length of the encoder); therefore, it is practically limited to short codes, usually $\nu \leq 8$. For sequential decoding, both the complexity and decoding effort are nearly independent of ν, but the number of computations that the decoder must perform to decode the received sequence is a random variable of Pareto distribution [2] which makes the decoding incomplete in finite time. There is always a small probability that certain error patterns will never be decoded or that erasures* will ocour when the decoding attempt is terminated.

Under Low Cost Electronics we are experimenting with a recently developed algorithm known as a multiple stack algorithm that combines elements of both maximum likelihood and sequential decoding. Basically, this algorithm progresses rapidly through the code tree to make a coarse sequential decoding estimate and then examines the vicinity of each questionable branch point to refine the estimate. The problem of stack overflow is reduced by organizing the memory in multiple stacks, appropriately ordered by a likelihood function metric, in which tentative decisions are made and then stored. With the proper choice of codes, the multiple stack algorithm should provide a superior alternative to either the maximum likelihood or strictly sequential decoding methods used alone. The new algorithm has been implemented in our laboratory with an 8-bit microprocessor in a facility permitting design tradeoffs and direct comparisons to be performed easily.

---

*<u>Erasure</u> - Failure of the decoder to reach a decision.

1.3 Scope

To implement the MSA on a microcomputer and test its speed and error performance in a noisy environment, the following assumptions, based on practical considerations, have been made:

(a) The information can be processed independently in blocks if the block lengths (k) are at least 4 or 5 times the constraint length (ν).

(b) Erasure probability should be less than 10^{-5}.

(c) The undetected bit error rate should be less than 10^{-4} for signal-to-noise ratios above 5.5 dB (i.e., binary symmetric channel error rates less than 3×10^{-2}).

This report also considers the practicality of real-time MSA decoding using the Zilog Z-80 computer. There is reason to believe that if a high-speed microprocessor such as the Z-80 or one of its successors is used and if an adequate buffer is available, actual on-line sequential decoding can be achieved at acceptable data and low error rates.

Since quantitative results are difficult to obtain analytically for convolutional decoding, simulations were performed. The results presented in this report are based entirely upon these simulations. Although a binary, additive white Gaussian noise channel was used in the simulations, many of the results are applicable to other random noise channels.

The following section describes the MSA and the software implementation of the MSA on the Zilog Z-80 microcomputer system organized for this purpose on Project 7010. Section III discusses

the selection and construction of fast-decodable codes which achieve both a low error probability and a minimal erasure probability. Section IV describes the parameter selection, and Section V states some important properties of the MSA. The performance of the MSA and comparison with the performance of other convolutional decoding algorithms, especially that of the Viterbi algorithm, are analyzed and discussed in Section VI. The final section presents conclusions and suggestions for further improvements and applications.

SECTION II

MULTIPLE STACK ALGORITHM

2.1 Brief Review of Convolutional Codes

Reliability, simplicity of implementation and nearly optimal performance make convolutional decoding by means of the MSA an appealing application example for effective use of standard microprocessors as signal processors. Figure 1 shows a very simple convolutional encoder. It consists of a 3-stage* shift register with information bits shifted in sequentially, two modulo-2 adders (exclusive OR's) and a multiplexer for the two resulting bit streams. In the terminology of convolutional codes, the encoding constraint length, ν, is equal to the number of shift register stages and the code rate is the number of bits k_o shifted into the register divided by the number of multiplexed output bits n_o sent on the channel. In the example, $\nu = 3$ and the code rate $(R_c) = 1/2$.

As previously noted, a convolutional code is conveniently illustrated by a tree diagram (Figure 2). The points of divergence are called nodes, the horizontal transitions are called branches. If the first input bit is a zero, the code symbols are those shown on the first upper branch, while if it is a one, the output code symbols are those shown on the first lower branch. Similarly, if the second input bit is a zero, we trace the diagram upward. In this manner, all sixteen possible outputs for the first four inputs may be traced.

Convolutionally encoded messages are considered reliable because a given number of encoded symbols $(n_o\nu)$ affects each information bit

*It is conventional [4] to regard the blocks in Figure 1 as shift registers and the intervals before and between as stages.

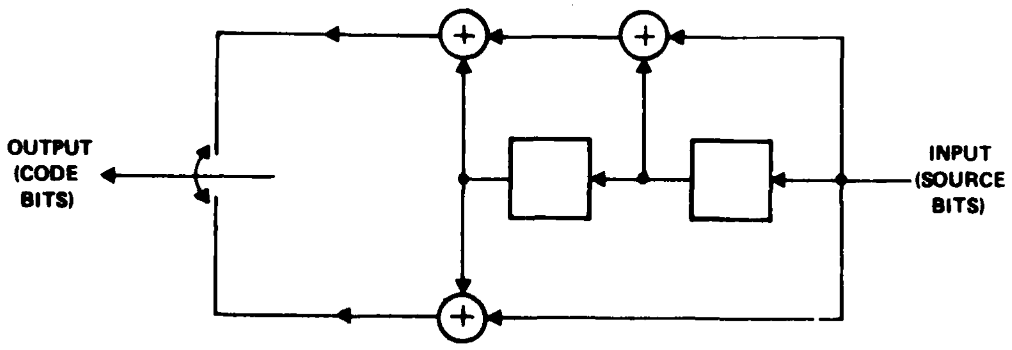


Figure 1. A SIMPLE (2, 1) 3 CONVOLUTIONAL ENCODER

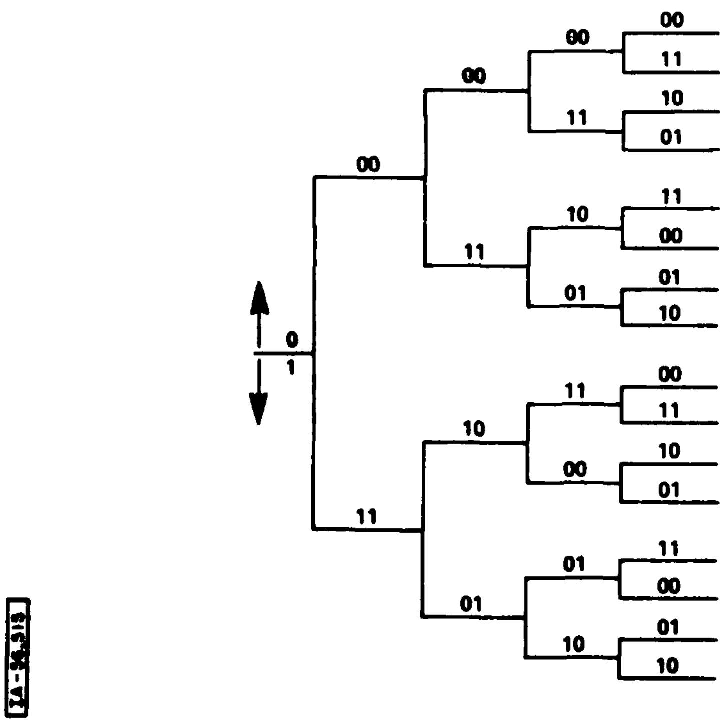


Figure 2. TREE CODE REPRESENTATION FOR THE (2, 1) 3 ENCODER OF FIGURE 1

permitting the possibility of correcting noisy data. For each information bit incorrectly transmitted, there will be $n_o\nu$ symbols to check and correct it.

These tree codes can be practically decoded by maximum likelihood decoding algorithms only when the constraint length is short. The Viterbi algorithm, for instance, combines nodes representing equivalent states to form a trellis diagram representation of the code tree, and systematically searches the trellis for maximum likelihood estimates of transmitted sequences. Only the most likely branch into each node at a given trellis depth is retained. But when the constraint length of the code becomes very large, the number of paths in the trellis becomes so great that this kind of decoding method is impractical.

The independent nodes create paths, the number of which increases exponentially with distance as we progress deeper into the trellis. But, since many different paths stem from the early nodes in the tree, a measurement possibility presents itself. If we could use a means of measuring the quality of the paths to the successive nodes, we might then be able to effectively disregard paths judged to be sufficiently bad.

Such a quality measure does exist in the form of the Fano metric, which is stated below:

$$M_f\ (r) = \log_2 \frac{P\left[\underline{Y}(1,rW) \mid \underline{X}(1,rW)\right]}{P\left[\underline{Y}\ (1,rW)\right]} - rWB \qquad (1)$$

where $\underline{Y}_{(1,\ rW)}$ and $\underline{X}_{(1,\ rW)}$ are the first rW digits respectively of the received sequence $\underline{Y}$ and the transmitted sequence $\underline{X}$. W represents

the number of code bits per source bit (i.e.; $1/R_c$) and r indexes the depth in the tree. B is a bias term which has been shown by J. L. Massey to be optimal (in the sense of minimizing decoded error probability) when it is nearly equal to the code rate (R_c) [3].

On the average, the Fano metric increases monotonically on the correct path and declines in pursuit of an incorrect path. The metric values are calculated by using prior knowledge (or assumption) of the channel error probability and the Hamming distance between the transmitted and received sequences. As the decoding algorithm progresses through the tree, the metric value must be stored in association with the corresponding paths.

2.2 The Single Stack Algorithm - A Sequential Decoding Method

Sequential decoding is a generic term for a tree code probabilistic decoding procedure that operates by making tentative hypotheses on successive branches to establish a path through the tree and by changing the path hypothesis when subsequent choices indicate that earlier choices were incorrect. The Fano algorithm represents one of the early methods of sequential decoding developed for a class of random tree codes [4]. This algorithm progresses either backward or forward through the code tree one node at a time according to the path metric value relative to a preset threshold.

Although the correct path is ultimately found, the search for that path is restricted to a route through connected nodes separated by a single branch, whether the decoder is advancing or retreating in the code tree. The Fano algorithm method tends to maintain decoder hardware complexity and storage requirements (excluding buffer storage) to the approximate level of the encoder. This is achieved, however, at the expense of many sequential computations during periods of high channel noise.

The single stack algorithm (SSA), developed by Zigangirov and later by Jelinek [5], is an improved sequential decoding method. The SSA reduces the complexity of path search by providing for storage and metric reordering of all previously processed path data. This allows the decoder to return to one of a number of previously explored nodes according to the relative path metric values appropriately ordered and stored in a memory stack. This improvement is achieved at the expense of increased memory requirements and the need to reorder at each node extension the path metrics stored. The SSA avoids repeating earlier computations.

The principal strengths of the single stack algorithm are:

(1) Bit error rates which decrease exponentially with code constraint length at information rates below channel capacity,

(2) A computation load that remains bounded, independent of constraint length, provided the code rate is less than a computational bound (R_{comp}),

(3) Storage requirements which grow only linearly with code constraint length.

The SSA also has weaknesses which could make its results unacceptable in high noise situations. One such limitation is the variability in computation time to advance one node in the tree. To keep up during noisy periods or to catch up quickly when the channel has again become quiet, the decoder is forced by this variability to include sufficient buffer storage and to possess a speed advantage relative to the channel data rate. It must also have sufficient storage capability in the memory stack.

We know that sequential decoding can be described quantitatively by a computational effort C. We interpret C to be the number of operations or computations required to advance a unit distance in the code tree. C is a random variable having a Pareto distribution*. In other words, the probability $P(C>N)$ that C exceeds some large value N is proportional to $N^{-\psi}$

$$P(C>N) = \alpha N^{-\psi} \tag{2}$$

where the Pareto exponent ψ is independent of constraint length, depending only on channel properties and code rate (α is a constant of proportionality) [2].

Since most received sequences at useful channel signal-to-noise ratios are decoded with very little effort, average computation should be lower than the fixed computational effort of maximum likelihood decoding. But there is always some fraction of received sequences proportional to $N^{-\psi}$ which imposes an impractically large computational load and results in incomplete decoding, which we interpret here as producing erasures.

Normally a computational limit (C_{lim}) is established beyond which the decoder fails, erasing a stream of data. The probability of such an erasure is governed by the Pareto distribution:

$$P_{ERASURE} = P(C>C_{lim}) = \alpha C_{lim}^{-\psi} \tag{3}$$

*Regardless of the algorithm used, sequential decoding involves a random motion in the tree, and hence C is a random variable.

which decreases algebraically as a function of C_{lim}.

These erasures are generally caused by an overflow of the available memory in the input buffer which stores continuously arriving channel data, or by overflow of the finite memory stack which stores and reorders the processed data. The latter occurrence is more prevalent. Figure 3 presents a flow chart of the SSA. The basic steps that comprise the SSA codes are:

INITIALIZE	(1)	Initialize by clearing the memory table and creating one entry corresponding to the start of the decoding tree.
ORDER	(2)	Retrieve the entry with the largest metric.
EXTEND NODES	(3)	Compute the branch metrics stemming from the node found in step (2) and create new entries in place of the original. Store these new entries in the table.
CHECK TERMINAL, CHECK C_{lim}, OUTPUT ERASURE OR DECISION	(4)	If the computational limit is exceeded, the procedure halts and causes erasures. If the C_{lim} is not exceeded, repeat steps (1) through (3) until a terminal node is reached and then read out the decoded path from the origin to the terminal node.

When the computational limit is exceeded (or the stack is full in the SSA sense), this algorithm just makes a random guess and produces erasures. This event is governed by the Pareto distribution as previously discussed.

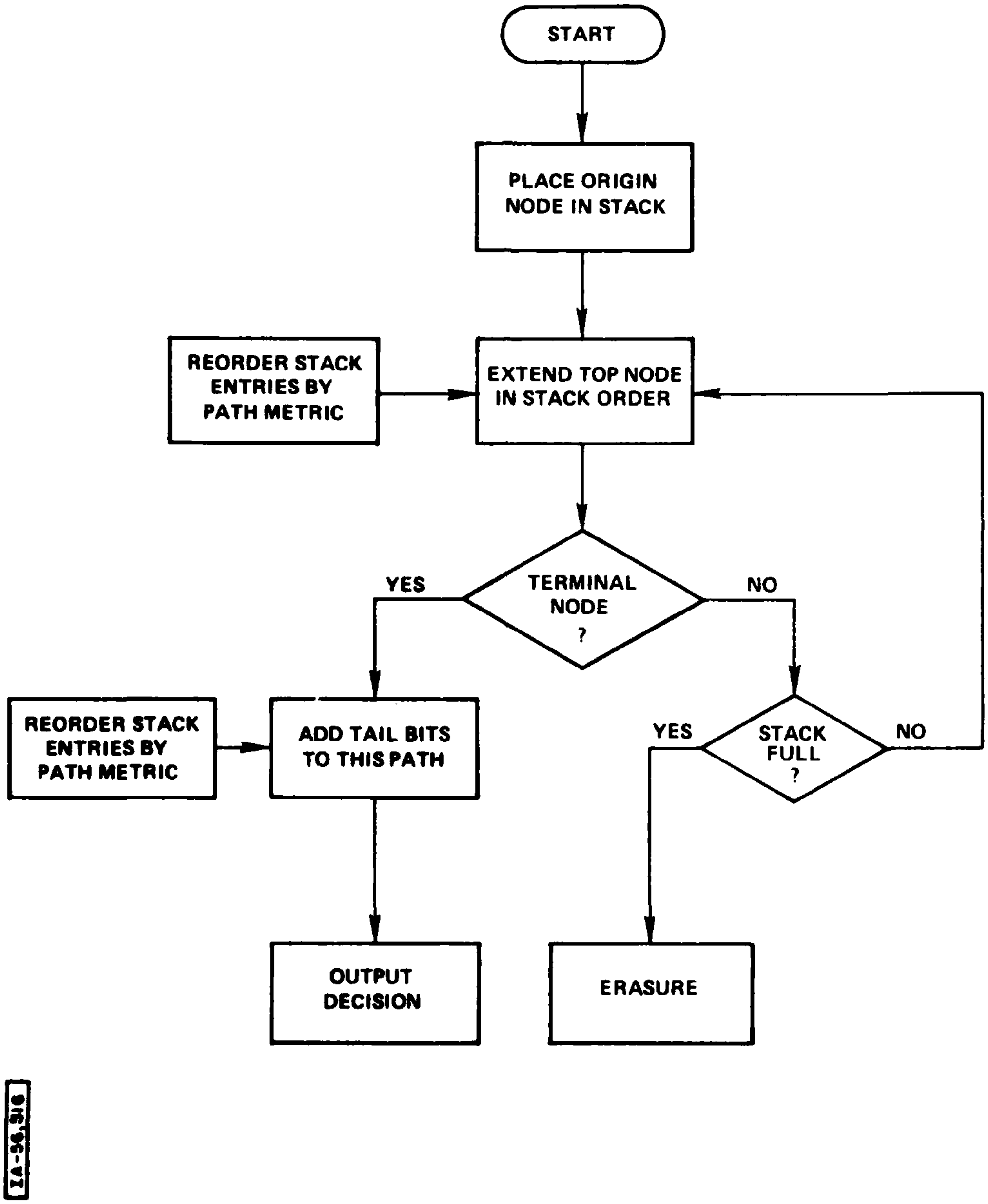


Figure 3. FLOWCHART OF THE SINGLE STACK ALGORITHM (WITH FINITE SIZE MEMORY STACK)

2.3 An Improved Stack Algorithm - The Multiple Stack Algorithm (MSA)

2.3.1 Basic Strategy of the MSA

The multiple stack algorithm (MSA) is a sequential decoding procedure devised by Chevillat and Costello to overcome the deficiencies of the single stack algorithm [1]. The MSA strategy differs from that of the SSA. Where the SSA advanced slowly during noisy periods exploring many incorrect subsets before extending the correct path, the MSA progresses quickly through the tree to find a reasonably good tentative estimate after which the alternatives are explored in search of the maximum-likelihood path.

The problem of stack overflow is accommodated by providing a hierarchy of memory stacks. Each stack is used to process a subset of nodes having the best metric values transferred to them from the previous stack. As these distributed stacks are filled, the data is processed in the final stack to arrive at a tentative decision for that stack. This decision is stored and compared with a new tentative decision reached by processing in the previous stack, and the processing is continued until a decision is reached in the first stack. The best tentative decision is kept at each iteration, and, when processing is concluded in the initial stack, the best decision becomes the decoded path. Decoding can also be terminated by exceeding the determined C_{lim}; in this case, the best tentative decision yet obtained becomes the decoding decision. The MSA produces the error performance of an SSA with infinite stack size by iterating the decoding algorithm in a set of subordinate, finite-size stacks.

Although it is Pareto distributed, the probability distribution of computational effort can be upper-bound by an exponentially decreasing

function of the code's column distance function*. This fact is useful for the MSA and makes it best suited for decoding codes with large column distance function.

The MSA retains these desirable properties of sequential decoding:

1. complexity and computational effort are relatively independent of code constraint length, and

2. average decoding is faster than maximum likelihood decoding.

Furthermore, the MSA allows complete erasurefree decoding. Such a complete decoding method, capable of achieving low error probabilities with substantially lower average decoding effort than the Viterbi algorithm, would seem to be a desirable compromise between direct maximum-likelihood and strict sequential decoding.

2.3.2 Flowchart Description of the MSA

The multiple stack decoder consists of a central processor and a number of finite-size memory stacks. Decoding begins by placing the origin node of the estimated code tree into the first stack. The flow chart in Figure 4 shows that the MSA initially operates exactly like the conventional single stack algorithm. Starting with the origin node, the top node in the stack is extended. After its elimination from the stack, the successors are inserted and the stack is put in order according to the metrics of the nodes. Each entry consists of, among other things, a node identifier and its metric. Decoding proceeds by extending the node on the top of the stack again. The terminal node distance is established by setting a

*Column distance function (CDF) is defined as the minimum Hamming distance between distinct paths divergent from the first branch as a function of distance into the code tree.

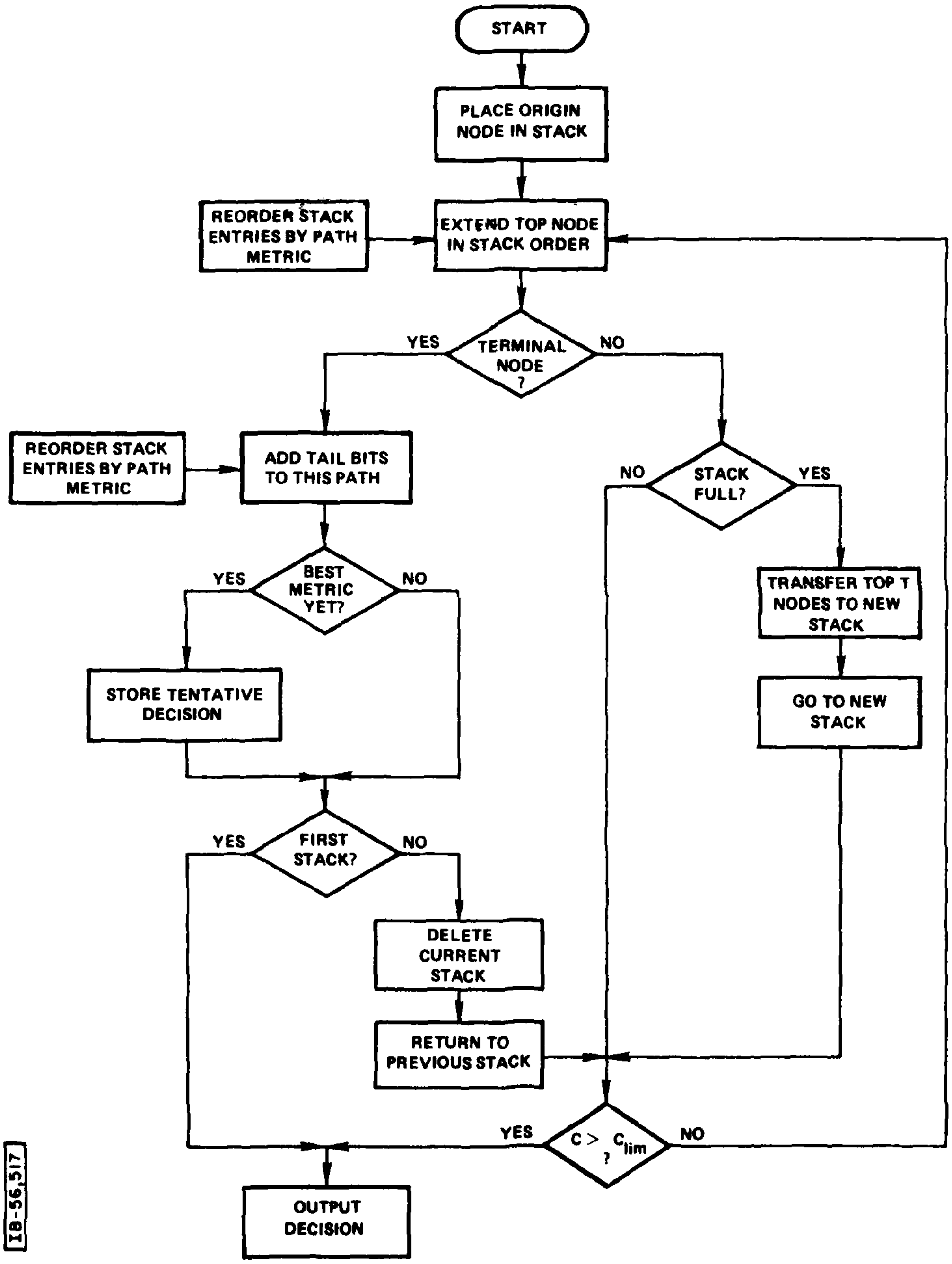


Figure 4. FLOWCHART OF THE MULTIPLE STACK ALGORITHM

decoder constraint length. It is normally 4 or 5 times the encoder constraint length, since all paths merge into the same nodal state with high probability at this approximate distance [6]. The distance of the terminal node determines the size of the processed blocks of data. The MSA processes each block as a unit.

The basic steps that comprise the MSA are:

<u>INITIALIZATION</u>

(1) Initialize by clearing the memory table and creating one entry corresponding to the start of the decoding tree.

<u>PATH EXTENSION/ORDERING</u>

(2) Retrieve the entry with the largest metric.

(3) Compute the new branch metrics stemming from the node found in step (2) and store these entries in place of the original.

<u>STACK CREATION</u>

(4) If the first stack table is full, transfer the top T entries into the next secondary stack and continue to process in subsequent stacks as necessary until a temporary decision is reached for a complete path containing k information bits, where k is set by the user.

<u>BACK SEARCHING (STACK DELETION)</u>

(5) Go back by processing in reverse order in the set of subordinate stacks to improve the decision by finding a complete path with a larger metric value to replace the current one until the computational limit is reached.

TERMINATION

(6) Terminate if (i) a decision is reached at the first stack, or

(ii) the computational limit C_{lim} is reached.

Notice the differences between the MSA and SSA. Instead of quitting when the first stack has filled and calling the entire block an erasure, the MSA continues processing in secondary stacks to reach a decision, backsearches to refine the tentative decision, and then terminates decoding. Termination occurs either after satisfactorily achieving maximum likelihood decoding or by reaching a set computational limit.

If the terminal node is reached before the first stack fills up, decoding is completed, and the path from the origin to this terminal node becomes the decoded codeword. In this case, the MSA executes exactly the same decoding steps as the SSA. If the first stack fills up before the terminal node has been reached, the top T nodes of the first stack (the most likely ones) are transferred to a second stack where decoding proceeds using these T transferred nodes. T is a decoding parameter to be selected.

If the top node in the second stack reaches the terminal node before the stack fills up, the codeword corresponding to this terminal node is stored as a tentative decision in a special register. The decoder then deletes the remaining nodes in the second stack and returns to the first stack where decoding continues. Since T nodes have been removed and transferred at the time of overflow, exactly T spaces are available in the first stack. If the decoder reaches a terminal node before the first stack fills up again, the path metrics of the new terminal node are compared with those of the tentative decision. The node with the best path metric becomes the decoding decision.

But, if the first stack fills up again before the terminal node is reached, a new second stack (the previous entries in the second stack having been deleted) is formed by again transferring the top T nodes of the first stack. If this stack also fills up before a tentative decision can be made, a third stack is formed by transferring the top T nodes from the second stack. Additional nodes are formed in a similar manner until a tentative decision is reached. The decoder always compares a new terminal node's path metric to that of the node stored in the tentative decision register and retains the node with the best metric available. The rest of the nodes in the stack are then deleted and decoding proceeds in the previous stack.

The algorithm terminates decoding when it reaches the end of the tree in the first stack or when it exceeds the C_{lim} set by the user. In the former case, the node at the top of the first stack becomes the final decoding decision (maximum-likelihood decoding). In the latter case, the best tentative decision yet obtained becomes the final decoding decision (non-maximum-likelihood decoding).

We would like to take a moment to discuss the possible occurrence of undetected (or undetectable) errors in the MSA decoder. Undetected errors will occur if the channel error patterns are such that:

(1) the correct path was not transferred to secondary stacks, and C_{lim} was reached before returning to the first stack (e.g., the correct path was available throughout the decoding process, but, because C_{lim} was reached, an incorrect path was elicited), or

(2) the correct path was transferred from the first stack, but the largest metric found for all decoding decisions at secondary stacks was equal to or greater than that for the correct path, or

(3) the metric value of the correct path was less than the value of some incorrect path such that the incorrect path with the larger metric value was finally preferred by the decoder, and decoding was completed in the first stack.

The error events of type (3) are simply those that would remain undetected by an ideal maximum-likelihood decoder. The error events of types (1) or (2) would be decoded by such a decoder. The parameters of the MSA should be chosen to minimize the events of types (1) or (2) at the required throughput.

2.3.3 Software Implementation of the MSA

The MSA was implemented on a Zilog Z-80 microprocessor in our laboratory. This versatile, third generation machine was utilized for several reasons including the fast instruction cycle time, operations-code efficiency, multibyte instruction capability, and suitability for the stack reordering operation. To satisfy the anticipated memory requirements of the MSA, a RAM containing 48 k bytes of memory was incorporated into the development system. Rate ½ convolutional codes of constraint length 8 and 15 were considered for implementation of the algorithm, as practical and challenging examples.

The entire simulation can be separated into three parts: the random error generator, the convolutional encoder and the MSA decoder, as shown in Figure 5.

First, a 16-bit linear feedback shift register pseudo-random number generator is used to provide sequences of nearly random source bits to be encoded by the $(2,1)\nu$ convolutional encoder. The resulting codewords are then fed to a simulated binary symmetric

18-56,518

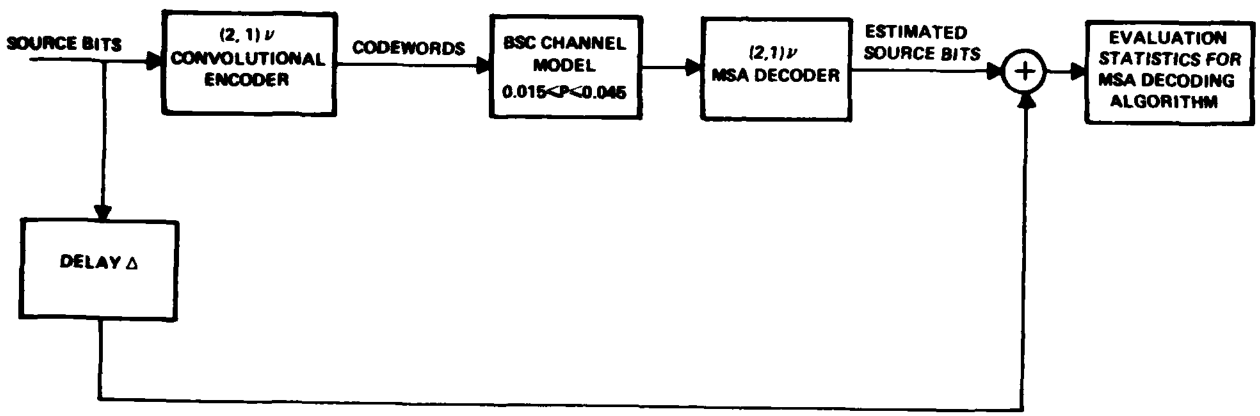


Figure 5. BLOCK DIAGRAM OF THE MULTIPLE STACK ALGORITHM SIMULATION

channel (BSC) which combines the output of a linear congruential random number generator* with the bits produced by the source generator. The inputs to the MSA decoder are the corrupted channel digits.

Finally, the decoded message sequences of the MSA are compared with the suitably delayed version of the actual transmitted message sequences**. The number of errors undetected by the MSA is finally recorded and serves as a measure of the performance of the MSA decoder. Details of these blocks of the MSA simulation are presented below.

2.3.3.1 <u>The Random Error Generator</u>. The random error generator, which is used to simulate the effects of additive white Gaussian noise on a communication channel, consists of a 16-bit linear feedback shift register (LFSR), pseudo-random number generator with feedback polynomial 210013_8 and a 16-bit linear congruential pseudo-random number generator which produces a sequence of residues of a large modulus by means of a linear transformation that uses the recursive relationship:

$$Y(i + 1) = (2^2 + 1)\, Y(i) + 1 \mod 2^{16} \qquad (4)$$

It can be shown that both congruential and shift-register generators have regularities that make them individually unsuitable for general Monte Carlo use, but combining the generators in various ways appears to produce satisfactorily random sequences. The method of combination

*A linear congruential generator produces a sequence of residues of a large modulus m by means of a linear transformation $x_{i+1}=ax_i+b$ mod m. $0 \le x_i < m$ [23].

**Normally, a decoding decision is withheld until the entire received sequence has been processed.

we used is known as algorithm M. It combines two sequences U and V of uniform distribution $U_1, U_2, U_3, \ldots . V_1, V_2, V_3, \ldots$ having magnitudes between 0 and $2^{16} - 1$, produced by congruential and shift-register generators. Suppose memory locations C(1), C(2) . . . . C(100) are filled with U. One generates a new V, uses its last seven digits to form an index J from 0 to 127 and uses C(J) with a newly generated U.

Sequences obtained by applying Algorithm M successfully passed the tests usually applied to uniform random number generators, and also passed the chi-square test of frequency distribution.

2.3.3.2 The Convolutional Encoder. Encode is our algorithm used to generate the digits associated with the tree branches at a rate required by the decoder. There are at least two approaches to convolutional encoding which differ in storage requirements.

A. Table - Lookup

For a simple (2,1)8 code, encoding can be achieved by table lookup methods which eliminate complicated repetitive computations. Simple simulations of the encoder operation can be accomplished by forming a table with 256 possible connections of 8-stage shift registers. The tables permanently store the information.

The basic tradeoff in the table-lookup method is time vs. memory. The required size of the codeword table, which increases exponentially with the constraint length of the code, obviously sets a limit to the type of encoder simulation. A table-lookup method is practical only for short constraint length codes, and these have limited error correction capability.

B. Software Emulation

For longer codes, such as (2,1)15 codes, it is better to emulate the actual encoder with a series of bit shifts, AND, and EXCLUSIVE-OR operations according to the given generator polynomials, generating codewords as required. These are extremely simple operations for the arithmetic logic unit of the Z-80's CPU and can be programmed efficiently. This method is also flexible because codes with different generator polynomials can be implemented on the microcomputer by a simple software change in the encoder tap connections.

2.3.3.3 The Multiple Stack Decoder. The multiple stack decoder block is the main program of the microprocessor simulation. The inputs from the MSA are sequences of corrupted received channel symbols. The decoder includes, among other things, a replica of the convolutional encoder. This encoder constructs a decoding tree with which the received channel symbols are compared to obtain the best estimate. The outputs of the MSA are the decoded message bits. The probability of an undetected error decreases exponentially with the size of the replicated encoder built into the decoder. In short, the MSA decoder utilizes the information of both the received symbols and the code tree replica to recover the actual transmitted bits.

2.4 Some General Merits of the MSA

2.4.1 Completeness

For those few received sequences which require excessive searching because of high noise, the MSA maintains a good non-maximum likelihood estimate of the correct codewords. It does not suffer from the deficiencies related to Pareto distribution of computational

effort which can cause incomplete decoding in the SSA in high noise environments. The MSA establishes several secondary stacks to process the decoding and to reach some tentative decision before exceeding the computational limit. Therefore, all codewords are decoded to some estimate within the computational limit. The completeness of MSA decoding is sometimes termed erasure free decoding.

2.4.2 Optimality

The MSA is a sequential decoding method which achieves asymptotically optimum error performance. Because the complexity of sequential decoders is relatively independent of constraint length ν, the constraint length can be made quite large ($24 \leq \nu \leq 48$) with a very small probability of undetected error. In the Z-80 implementation, the constraint length is restricted to a smaller number because of limitation on memory and processor speed. For rates above R_{comp}* the error performance of the sequential decoding algorithm has been shown elsewhere to be superior to that of certain block codes of the same length [7]. The MSA also bears certain similarities to that of the Viterbi algorithm particularly the ability to do some limited multiple-path extension.

2.4.3 Decoding Efficiency

The decoding effort in the MSA is characterized by a random variable. It allows some adaptability to the channel noise environment and is faster on the average than the constant decoding rate of Viterbi decoding which has a fixed computational effort regardless of the channel environment. The MSA operating in real-time need never be idle [5]. If the most recent received signals correspond

*The computational cutoff rate of sequential decoding above which the average number of computations per information bit tends to infinity for large block size.

to depth d in the decoding tree, and if the current node is at depth d, then, while waiting for more received signals, the decoder can extend the stack entry which has the largest metric and which is at a depth less than d. This advance work costs nothing, and since it may be required at a later time, it permits efficient operation at high data rates.

Such a complete decoding algorithm capable of achieving low error probabilities with a lower average decoding effort at higher throughput may be a desirable alternative to the popular Viterbi maximum-likelihood decoder in various applications. Although not explored here, it also lends itself to soft decision decoding. It could provide an effective inner code in a concatenated coding scheme [8].

2.5 Codes Used in the MSA

It has been shown that codes with certain distance properties will provide both low error probabilities and fast decoding for the MSA [9]. These properties are closely related to the column distance functions of the codes. Guided by this function, searches for good codes suitable for the MSA were performed and will be reported in the next Section.

SECTION III

CODE SELECTION FOR THE MSA

3.1 Systematic Versus Nonsystematic Codes

An (n_o, k_o) convolutional code is called systematic if a fixed set k_o of the output symbols is equal to the current k_o input symbols. Otherwise, the code is nonsystematic.

In the absence of errors, the use of systematic codes provides immediately apparent information sequences from the encoded sequences. For many engineering purposes, such as synchronization and monitoring, it is desirable to get reasonably good estimates of the information digits directly from the received sequences without first employing the decoding process. In nonsystematic codes, however, the information symbols are diffused by linear combination, and this convenience is unavailable.

Wozencraft and Reiffen have shown that for any nonsystematic code there is a systematic code with the same minimum distance [10]. Their results showed no advantages in using nonsystematic codes with threshold decoding, but their research may have overlooked advantages of using nonsystematic codes with other types of decoders not then available.

Recent results have demonstrated reduced undetected decoding probability by use of nonsystematic codes with either sequential or maximum likelihood decoders [11]. It has been concluded further that the nonsystematic codes have simpler encoder realizations (shorter encoding constraint length) than the equivalent systematic codes. For these reasons, nonsystematic codes were chosen over systematic codes for our MSA sequential decoder implementation.

3.2 Best Codes for the MSA

3.2.1 The Free Distance Consideration of the MSA

For many applications, the code rate $R_c = \frac{1}{2}$ is chosen as a compromise between bandwidth expansion and correction capability. The resultant doubling of bandwidth attains within 1 dB the total gain possible by this type of coding, but does not catastrophically degrade the energy per transmitted source bit. A new coding gain remains. Coding gain analyses previously performed on this subject have also shown $\frac{1}{2}$-rate codes to be optimally efficient on additive Gaussian noise channels [12].

For our study, the rate $\frac{1}{2}$ Bahl-Jelinek (B-J) nonsystematic codes with complementary generator were chosen to obtain the largest free distance [13]. The free distance of a linear convolutional code is the minimum distance between pairs of infinite length code sequences; it is also equal to the minimum Hamming weight of the sequence. Of all convolutional codes known so far, B-J codes are the only ones that achieve maximum free distance:

$$d_{free} = \nu_o + 2 \text{ for } \nu_o \leq 16 \tag{5}$$

where ν_o is the encoding constraint length of the B-J codes.

3.2.2 The Column Distance Function Consideration for the MSA

To choose the best available codes for the MSA, the relationship between the column distance function and the computation effort of the MSA has been studied. The column distance function ($d_c(r)$) is the minimum distance between code sequences that diverge from the first branch as a function of depth r in the tree. The column distance function is bounded at the constraint length distance by the

minimum distance (the distance between sequences at a depth equal to the constraint length) and by the free distance

$$d_c(\nu) \leq d_c(r) \leq d_c(r \to \infty); \; r \geq \nu \tag{6}$$

Chevillat and Costello have shown that the computational effort required for sequential decoding of a convolutional code is upper bounded by an exponentially decreasing function of the code's column distance function [9]. Codes with rapidly increasing column distance function are best suited for the MSA.

3.2.3 Code Selection Procedure

Table I and Table II shown below have been generated by Chevillat to enable the selection of good codes for the MSA [14]. We have used these tables for our code selection as will be described. Table I displays an "influence limit" parameter d_{max} as a function of the crossover probability for a binary-symmetric channel. This parameter (d_{max}) is defined as the value of the code's column distance function beyond which it has little or no influence on the Pareto distribution of computational effort, $P(C>N)$. For a given expected channel crossover probability (or equivalent signal-to-noise ratio) we would like to choose a code whose CDF reaches at least d_{max} in order to control $P(C>C_{lim})$.

In addition, we also should choose a code having large free distance, d_{free}, in order to minimize the residual error probability P_E. To do this we attempt to reach the condition of equation (5) for the Bahl-Jelinek nonsystematic codes.

For the MSA we also need to select codes with rapidly increasing column distance function to control the computational load. Chevillat

has presented (in Table II below) the generator polynomials for codes having optimum average column distance growth as a function of d_{max} and the code's free distance, d_{free}.

For a (2,1)8 code we choose $d_{free} = \nu + 2 = 10$. For a typical channel crossover probability, $p = .03$, we see from Table I that $d_{max} = 10$ also; that value is also adequate for $p < .03$. We select from Table II a code, with generator polynomial $(1344)_8$, that meets the distance requirements.

Similarly, we can choose a code having longer constraint length for use on noisier channels. For a channel crossover probability of $p = .048$, we find in Table I that $d_{max} = 15$. For a constraint length $\nu = 15$ code, we can obtain $d_{free} = \nu + 2 = 17$. From Table II we find that a (2,1)15 code having generator polynomial $(141512)_8$ has rapid column distance growth and satisfies the distance requirements.

We can construct both the (2,1)8 and (2,1)15 encoders using the generator polynomials selected above. As stated earlier, these codes are Bahl-Jelinek (B-J) nonsystematic codes with complementary generators. The selected generator polynomial is therefore complemented to yield another polynomial in such a way that their first and last coefficients are "one" and the middle coefficients are complementary. The resulting two polynomials are then used as the tap connections of the two shift registers which generate the code by generating two output bits for each input source bit. For a (2,1)8 code, the chosen generator polynomial is $(1344)_8$. The complementary generator polynomial is then $(1434)_8$. The (2,1)8 B-J encoder is constructed as shown in Figure 6. Similarly, the generator chosen for the (2,1)15 B-J code is $(141512)_8$. The

TABLE I

Influence Limit of CDF on P(C>N)

BSC-p	d_{max}
0.001	4
0.004	5
0.008	6
0.013	7
0.019	8
0.025	9
0.030	10
0.034	11
0.038	12
0.042	13
0.045	14
0.048	15

TABLE II

Optimum Average Column Distance Growth Codes ($6 \leq d_{free} \leq 11$)

d_{free}	d_{max}							
	4	5	6	7	8	9	10	11
6	15	15	15					
7	144	144	144	144				
8	122 142	122 142	142	142	142			
9	121 145 123 131	121 123	121	121	121	121		
10	1324 1264 1544 1344	1544 1344	1544 1344	1544 1344 1154	1344	1344	1344	
11	1542 1262 1512 1226 1442 1422 1412 1206	1522 1226 1422 1412 1206	1206	1206	1206	1206	1206	1206

TABLE II (Continued)

Optimum Average Column Distance Growth Codes ($12 \leq d_{free} \leq 14$)

d_{free}	d_{max}									
	5	6	7	8	9	10	11	12	13	14
12	1541 1225 1355 1365 1571 1431 1415 1231 1213 1425	1571	1571	1365	1365	1365	1541	1541		
13	12064 12244	12064	12064	12064 12244	12244	12244	11724	11724	11724	
14		12062	13732 12242	12242 12346 14122 13732 13542	12242 13732	12242	14112 12242	11026	11026	11026

TABLE II (Continued)

Optimum Average Column Distance Growth Codes ($15 \leq d_{free} \leq 18$)

d_{free}	d_{max}										
	6	7	8	9	10	11	12	13	14	15	16
15	12043 12055	12055	12055	12055	11451	11451 12055	14105	14105	14105	14105	14105
16	120564 120654 120714 120214 157304	120564 120714	120564	120564	163204	163204	141104	141104	141104	141104	141104
17	123722 122606	123722 122606	123722	123722	123722 141512	141512	141512	141512	141512	141512	141512
18	120645	155371	155371	133035	127611 133035	127611	127611	127611 137531	127611 137531 122145	127611 122145	122145

complementary generator becomes $(136266)_8$. The (2,1)15 encoder is shown in Figure 7. These encoders are used to generate our (2,1)8 and (2,1)15 code sequences for the MSA simulations.

IB-56,319

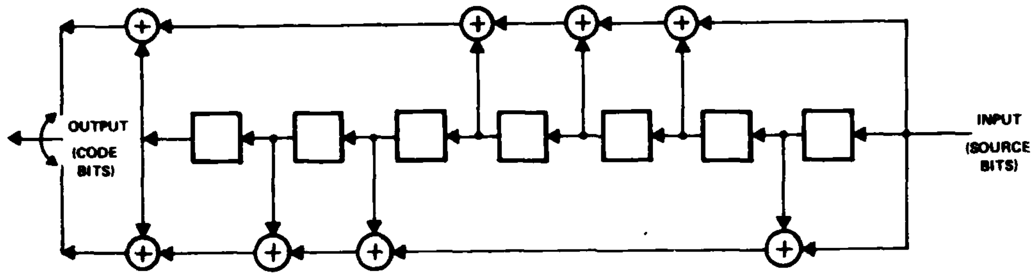


Figure 6. A (2,1) 8 CONVOLUTIONAL ENCODER WITH GENERATOR POLYNOMIALS $(1344)_8$ AND $(1434)_8$

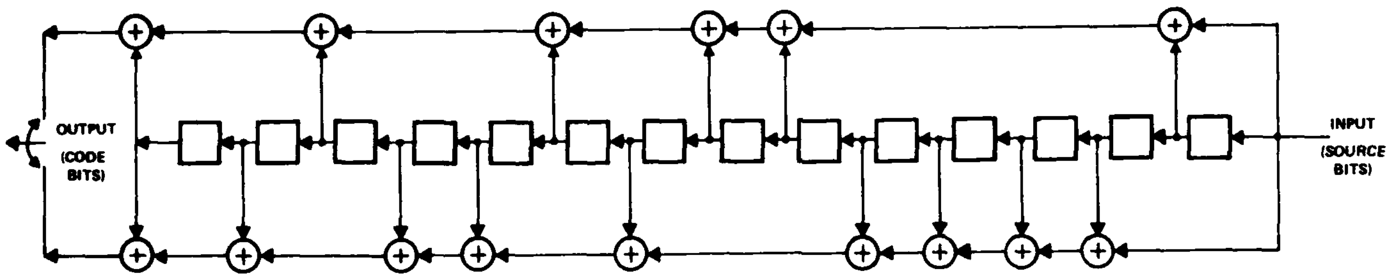


Figure 7. A (2,1) 15 CONVOLUTIONAL ENCODER WITH GENERATOR POLYNOMIAL $(141512)_8$ AND $(136266)_8$

SECTION IV

PARAMETER SELECTION FOR THE MSA IMPLEMENTED ON A Z-80 MICROCOMPUTER SYSTEM

Parameter selection for MSA use with the Z-80 is an important consideration in the attainment of low error rates and high throughput. Unlike most convolutional decoding algorithms, the MSA allows the user freedom to choose a strong constraint on the amount of memory used by adjusting the decoding parameters within a tolerable range of performance. Another user might consider the memory cost to be low enough to be willing to use as much memory as necessary to achieve high performance and throughput. This section points out various options available to the user in parameter selection and also considers those parameters that dominate MSA performance. It also discusses the impact of the Z-80 microprocessor implementation on parameter selection in terms of cost and effectiveness. Finally, an efficient way of choosing the parameter set is suggested.

To evaluate MSA performance in terms of its residual error probability (P_E) we must study P_E as a function of the MSA and channel parameters. In general, P_E increases as the channel signal-to-noise ratio (SNR) decreases. For MSA operation, the designer should adjust the decoder performance to accommodate variations in channel SNR by choosing the available number of stacks in order to keep $P(C>C_{lim})$ sufficiently small and by selecting the code constraint length (within computational limits) to minimize the residual error probability. While larger constraint length (ν) would ultimately improve error performance (exponentially) with only algebraic increase in complexity of the MSA decoder, only (2,1)8 and (2,1)15 codes were implemented and tested on our microcomputer development system because of storage limits.

To study the influences on performance of the channel crossover probability, memory stack size, selected computational limit, number of examined tree branches, and number of transferred nodes, extensive decoding trials were run during the Z-80 simulation. We expected the performance of the MSA to be sensitive to the restrictions imposed by a microcomputer system. Had we implemented the MSA on a fast minicomputer [14], we would have selected certain parameters to optimize the error performance and achieve erasurefree decoding with little regard for storage and computational effort. But for Z-80 implementation, the performance was restricted by available memory. The freedom to select parameters was similarly restricted.

We have available the following parameters in the MSA implementation:

- p: The crossover (transition) probability of the binary symmetric channel (determined for a white gaussian additive channel),
- Z_1: The size of the first stack,
- k: The number of branches through the tree from the root to terminal node (without tailing), it is also called the decoding constraint length,
- Z_i: The size of the secondary stacks $i = 2, \ldots h-1$,
- T: The number of nodes transferred from the previous stack upon stack overflow (the number of transferred nodes is the same for each transfer),
- C_{lim}: The computational limit beyond which the algorithm must terminate.

These six parameters and their effect on P_E will now be individually examined.

4.1 The Effect of Channel Crossover Probability p upon P_E

The transition diagram for the binary symmetric channel is shown in Figure 8. Consider the binary symmetric channel model with convolutional coding and MSA decoding shown in Figure 9. The encoded random sequence is corrupted by the additive white gaussian noise. Antipodal signalling and symmetric two-level received quantization are assumed. The effects of the transmitter, modulator, the gaussian channel noise and the receiver quantizer are all described by the BSC transition probability, which for the model of Figure 9 is

$$p = Q\left(\sqrt{\frac{2E_b R_c}{N_o}}\right) \tag{7}$$

where $Q(\cdot)$ is a normal distribution probability function in which E_b/N_o is the energy-to-noise ratio per source bit and $R_c = k_o/n_o$ is the code rate in bits per transmitted symbol. For all the codes we use here $R_c = \frac{1}{2}$. The probability function

$$Q(\alpha) = \frac{1}{\sqrt{2\pi}} \int_{\alpha}^{\infty} e^{-\alpha^2/2}\, d\alpha \tag{8}$$

is a well-tabulated function; the corresponding values of $(E_b/N_o)_{dB}$ for different values of p are shown in Table III.

For time invariant channels with additive white gaussian noise, the error performance without coding is upper-bounded by an exponentially decreasing function of signal-to-noise ratio [15]. With convolutional

IA-56,520

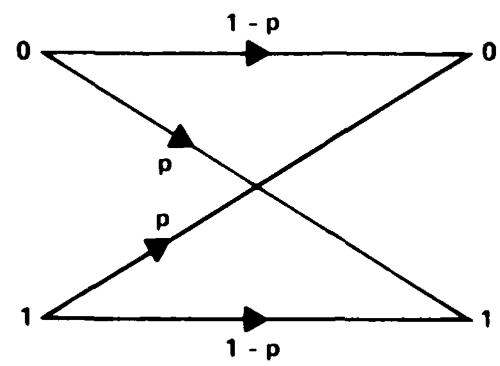


Figure 8. TRANSITION DIAGRAM FOR BINARY SYMMETRIC CHANNEL

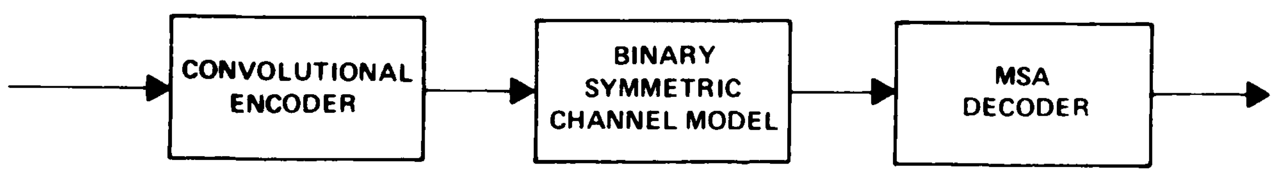


Figure 9. BSC MODEL WITH CONVOLUTIONAL CODING AND MSA DECODING SCHEME

TABLE III

The Corresponding E_b/N_o for Selected Cross-over Probabilities, p.

BSC "p"	$\left(\frac{E_b}{N_o}\right)$ dB
0.0449	4.609
0.0410	4.812
0.0371	5.032
0.0332	5.272
0.0293	5.529
0.0254	5.823
0.0215	6.129
0.0195	6.298
0.0176	6.465
0.0156	6.669

encoding and multiple stack decoding, the plot of decoded error rate P_E as a function of E_b/N_o is expected to decline much more steeply than the channel bit error rate curve of the BSC, provided E_b/N_o is sufficiently high.

But as the channel noise increases and E_b/N_o decreases, the error rate obtained by use of the MSA increases to a limit at which the deteriorating signal-to-noise ratio makes further coding undesirable as it would actually produce a loss rather than gain. For our simulations, the selected value of p is set by randomly toggling a fixed average number of error bits (according to a gaussian distribution) throughout all codeword bits being processed.

4.2 The Effect of First Stack Size Z_1

The first stack size should be large enough to completely process most of the received sequences so that only those sequences that are badly corrupted require the use of secondary stacks. The residual error probability of the MSA is upper-bounded by the decoded error probability of the single stack algorithm and other factors:

$$P_{E(MSA)} \leq P_{E(SSA)} + P_{es} \cdot P_1 \tag{9}$$

where $P_{E(SSA)}$ is the decoded error probability of an infinite-stack SSA (without erasures), P_1 is the probability of first stack overflow (causing erasures in the SSA) which is given by the Pareto distribution:

$$P_1 = P\left[C > Z_1 - 1\right] = C_{SD} \cdot \left[Z_1 - 1\right]^{-\psi} \tag{10}$$

and P_{es} is the erasure-estimate error probability (i.e., the probability of incorrectly estimating a bit in the codewords selected when $C>C_{lim}$). For example, the simple technique that picks at random a codeword whenever $C>C_{lim}$ has the probability $1 - 2^{-k} \approx 1$ of choosing the wrong path; in this case P_{es} depends on which path is chosen. A coin-toss estimate of the bits would produce $P_{es} = ½$.

The erasure-estimate error probability, P_{es}, is relatively small for the MSA because the entries with the best metrics are always included in the transfer of T entries to secondary stacks. Therefore, the component of $P_{E(MSA)}$ that is proportional to P_1 should decrease with increasing first stack size. If either P_{es} tends to zero (best estimate) or P_1 tends to zero (infinite first stack size), then

$$P_{E(MSA)} \leq P_{E(SSA)\infty} \tag{11}$$

which is the limiting case that achieves ideal performance.

Since $P_{E(MSA)}$ is influenced strongly by Z_1 (the size of the first stack), in principle we could have increased this parameter indefinitely to eliminate the second term in equation (9). Since this is impractical (it would degenerate to an infinite-stack SSA), we must vary the remaining parameters to control P_{es} and the computational effort to achieve efficiently a desired low error rate.

The first stack size (Z_1) has a strong influence on the overall error performance. Z_1 was made as large as practical in our simulation to allow a larger percentage of codewords to be decoded in the first stack which accomplishes maximum-likelihood (optimum) decoding. For those more noisy blocks, which overflow the first stack, the parameter

of computational limit enters into the picture. With a large Z_1, there are only a small portion of the codewords (actually proportional to $Z_1^{-\psi}$) that need secondary stack processing. C_{1im} is chosen to terminate multiple-stack processing after some reasonable time.

For cases like these, the MSA is designed to have at least one decision using multiple-stack processing. However, the performance can seldom compete with the optimum decoding of the hypothetical SSA with infinite stack size. Therefore, our first goal was to choose Z_1 as large as possible under practical memory constraints. This choice allowed the majority of codewords to have maximum-likelihood estimates. Later, C_{1im} will be chosen to yield a reasonable decoding time limit for remaining patterns needing additional processing. Theoretically, by making C_{1im} sufficiently large, decoding can be finished eventually after returning to the first stack.

Simulations we made have shown that by setting $Z_1 = 1024$ for (2,1)8 codes and $Z_1 = 2900$ for (2,1)15 codes the first stack size is large enough to obtain low error probability but without occupying an excessive number of memory locations in the microprocessor system.

4.3 The Effect of Decoding Constraint Length k

The decoding rate R_d is defined as:

$$R_d = \frac{k}{n_d} \tag{12}$$

where:

k: the decoding constraint length* or the number of information bits being decoded without any tail within each frame;

---

*k is also termed "decoding delay" in the sense that the decoder does not make any decisions until all k information bits have been decoded.

n_d: the total codeword bits (including tail bits) decoded for each frame.

A frame contains a total number of k information bits so that for a rate ½ code

$$n_d = 2 \cdot (k + \nu - 1) \tag{13}$$

where $\nu - 1$ is the number of tail bits added to the sequence. After substituting equation (13) into equation (12), we obtain

$$R_d = \tfrac{1}{2} \cdot \frac{k}{(k + \nu - 1)} \tag{14}$$

If k is made much larger than the encoding constraint length, the decoder rate remains at effectively the code rate of ½. The value of k does not greatly influence decoder memory or computational requirements, but as k is increased a decoding decision is deferred for a longer time and more bytes are required for each entry. More buffer storage and computation effort are required for larger blocks of k information bits. There is a connection between k and computational limit C_{lim} for fixed available memory. As k becomes larger, C_{lim} should be set smaller such that memory requirements are kept within the range of availability. P_E was shown to increase as k increased [16]. It has been shown that for a Viterbi decoder, a value of k that is 4 or 5 times the encoding constraint length ν is sufficient for negligible degradation from optimum decoder performance [17]. For the Z-80 implementation, we have bounded $k \leq 256$, and, in general, we choose $64 \leq k \leq 128$ for most situations.

4.4 The Effect of Size of Higher-Order Stacks Upon P_E

The higher-order stacks act as a supplementary aid for those received sequences which require searching in excess of the first stack. The use of large secondary stacks allows the MSA to reach the first tentative decision with fewer stacks, but the computational effort may be large. These two effects offset each other. P_E is independent of the choice of Z_i which only affects decoding time and computational effort. Z_i is normally much smaller than Z_1. For both (2,1)8 and (2,1) 15 codes, we have chosen $Z_i = 11$ for all $i \neq 1$; it was selected as a compromise between the number of stacks and the computational effort. With the higher-order stacks substantially smaller than the first stack, the error rate performance of the MSA approximates that of the SSA without erasures.

4.5 The Effect of T Upon P_E

An increase in the number of nodes transferred from the first stack requires more stacks to be formed and larger computational effort to reach the first tentative decision. But it also increases the probability that the correct node will be transferred to the next stack. These two effects offset each other. Again, P_E is independent of T, but T influences decoding time and computational complexity. In order to limit the required number of stacks and the computational effort, T was selected as $T \leq 4$.

4.6 The Effect of Computational Limit

An increase in computational limit allows more chance of decoding in the first stack rather than algorithm termination by exceeding a smaller C_{lim}. Any termination of decoding in the first stack implies an estimate at least as good as that of the infinite-stack SSA (it is

also a maximum-likelihood estimate). Therefore, P_E decreases with increasing C_{lim}. To ensure that the MSA is erasurefree, the decoder must have at least one tentative decision before decoding stops by reaching C_{lim}. For T = 1, a critical value of computational effort C_{crit} is defined, below which a tentative decision cannot be obtained. C_{lim} must be larger than C_{crit} to guarantee erasurefree decoding:

$$C_{lim} > C_{crit} \tag{15}$$

where

$$C_{crit} = \sum_{i=1}^{k-1} (Z_i - 1) + 2 (\nu - 1) \tag{16}$$

and ν is the encoding constraint length.

From Equation (16) we conclude that the computational limit C_{lim} and stack sizes (Z_1 and Z_i) are closely interrelated. If for example $Z_1 = 1024$ and $Z_i = 11$, C_{lim} must be at least greater than 1657 for the (2,1)8 MSA to ensure erasurefree decoding.

An increase of C_{lim} beyond C_{crit} does not improve P_E greatly. Therefore, we chose C_{lim} also to limit memory requirements. For Z-80 implementation, the saving of storage is a major concern and C_{lim} should be kept at a value that prevents memory overflow. A safe value of C_{lim} for the above (2,1)8 MSA case is 1700. For (2,1)15 MSA, we chose $C_{lim} = 3600$.

For T > 1, we expect that more computations will be required to achieve erasurefree decoding than for T = 1. Care was taken to select C_{lim} to be large enough to ensure erasurefree decoding.

4.7 Summary

In conclusion, Z_1 and k should be made large enough to satisfy the error probability required -- $P_E \leq 10^{-4}$. C_{lim} should be chosen to guarantee complete decoding without unduly stressing memory requirements. Z_i and T should be bounded to limit the required memory and the computation effort. A summary of our parameter selection for the (2,1)8 and (2,1)15 codes is shown in Table IV.

TABLE IV

Selected Parameter Values for (2,1)8 and (2,1)15 MSA

MSA Parameters	(2,1)8 MSA	(2,1)15 MSA
Z_1	1024	2900
Z_i	11	11
k	64	64
C_{lim}	1700	3600
T	3	3

SECTION V

SOME PROPERTIES OF THE MSA

Five properties of the MSA decoding are discussed below. Most of these properties and their proofs are taken from the work of Chevillat [14]. We include them here, with additional discussion, for the sake of completeness and to enable the reader to grasp more fully the salient features of the algorithm.

5.1 Properties and Their Proofs

Property 1: If the MSA terminates decoding by reaching the end of the tree in the first stack, its final decision is at least as good as the decision of a single stack algorithm with an infinitely large stack. That final decision is equivalent to a maximum-likelihood decoding.

Proof: If the MSA completes decoding in the first stack without forming any additional stacks, it executes the same steps as the SSA, and the decoding path (denoted here as j) will be the same as that of the SSA with an infinitely large stack.

If the MSA forms higher-order stacks to reach the first tentative decision and j is included in the T transferred nodes, j will be among the tentative decisions reached before returning to the first stack. The MSA's final decision will be at least as good as j.

If the MSA forms higher-order stacks to reach a tentative decision but j is not transferred to the next stack, the MSA obtains some tentative decision before returning to the first stack. Again the MSA's final decision will be made in the first stack and will be at least as good as j.

In the case enumerated in the proof, the correct path j is always among the decision elements, and the MSA's final decision must be at least as good as that of the SSA. Therefore, efforts are made to finish MSA decoding in the first stack to achieve maximum-likelihood decoding. It has been shown that the number of computations which the SSA decoder must perform to decode the received digit is a random variable of Pareto distribution; i.e.:

$$P\left[C > C_1\right] = C_{SA}\, C_1^{-\psi} \tag{17}$$

where C is the number of computations which the SSA must perform to decode a block of k information bits and ψ is a parameter which depends on the channel properties and the rate of the code. In the SSA, we call the probability of this single stack overflow the erasure probability. For the MSA, this is just the probability that the number of node extensions C (or equivalently the number of computations) equals or exceeds the size of the first stack Z_1, or $P\left[C > (Z_1 - 1)\right]$. According to Equation (17), this probability equals $C_{SA}\,(Z_1 - 1)^{-\psi} = P_1$. The proportionality constant C_{SA} and the parameter ψ are both independent of Z_1. (A detailed derivation of C_{SA} is contained in Reference [18]). Although P_1 can be made very small by making Z_1 very large, some fraction of codewords (even though very small) will always need secondary stacks to complete their decoding process. In the SSA, stack overflow causes erasures which might be intolerable for some situations. In the MSA, more stacks are available to obtain some decision before the computational effort is exceeded, as described in Property 2.

Property 2: In a noisy environment, received sequences which require excessive searching will cause the SSA to quit when stacks overflow. With the MSA, a tentative decision is always obtained before the computational limit is reached. Thus, while the SSA suffers from erasures (which means it would have to guess through the tree after the decoder quits), the MSA continues to decode and obtains a non-maximum likelihood decision which has a lower error probability than a random guess.

<u>Proof</u>: Because the MSA has additional stacks available for processing, a tentative decision will be reached if the computational limit is set sufficiently high. This temporary fast searching to get at least some results is a compromise between sequential searching and parallel processing. If the SSA produces an erasure, the user may (1) randomly select a codeword, or (2) make a coin-tossing guess to select the decoded bits. Since the decoding metric increases monotonically on the correct path and declines on all incorrect paths, and since the best metric examined is retained for a tentative decision, strategy (1) will, on the average, offer a path with a poorer metric than the tentative decision. In strategy (2), the path metric for the random guess is no larger than the smallest tree-path metric, since the decoded bits selected by the user are statistically independent of the source bits. Consequently, once a tentative decision is obtained, the estimate is better than a random guess.

Property 3: For $T = 1$, the number of stacks (u) formed before the terminal node is reached for the first time never exceeds the number of tree branches (k) without the tail (i.e., $P(u>k) = 0$ provided C_{lim} is not exceeded). At most, k stacks are formed to reach the first tentative decision.

Proof: We are trying to predict the maximum number of stacks required before a tentative decision is reached. For $T = 1$, only one node entry is transferred to a new stack at the time of overflow. The first stack guarantees that the top of stack 1 at the time of overflow has at least tree path length one. Similarly, the path at the top of stack i at the time of its overflow has at least length i. Since the tree does not branch in the tail and the tree is of path length k, to this point there will be a maximum of k stacks formed before the first tentative decision is reached.

This does not mean that we need only k stacks to terminate decoding in all cases. Although it requires a maximum of k stacks to reach the first tentative decision, the decoder may need more stacks to finish searching and comparison to reach the best decision. Property 3 only is true for $T = 1$; for $T > 1$, more than k stacks may be needed to reach the first tentative decision.

Property 4: For $T = 1$ and a code of rate ½ having constraint length ν, the MSA is erasurefree if C_{lim} exceeds a critical lower bound C_{crit} given by:

$$C_{crit} = \sum_{i=1}^{k-1} (Z_i - 1) + 2(\nu - 1) \qquad (18)$$

where Z_i, $i=1, \ldots C_{lim}$ is the size of stack i and k is the number of tree branches without the tail.

Proof: From Property 3 we learned that a maximum number of k stacks is formed before the first tentative decision is reached. Since Z_i-1 computations are executed in stack i before overflow

(where Z_i is the number of entries in the i^{th} stack, i=1, 2, . . ., k-1), a total of

$$\sum_{i=1}^{k-1} Z_i - 1 \tag{19}$$

computations are performed before the k^{th} stack is formed. In this last stack where the tree branches only once more before the tail, a maximum of $2(\nu-1)$ node extensions are executed before the end of the tree is reached (there are $(\nu-1)$ nodes to be branched into 2 possible tree paths). Hence, a maximum of

$$C_{crit} = \sum_{i=1}^{k-1} (Z_i - 1) + 2(\nu-1) \tag{20}$$

computations is necessary to reach the first tentative decision. As noted in Property 2, the MSA is erasurefree when at least one tentative decision is obtained before the decoding stops by reaching C_{lim}. If we set $C_{lim} \geq C_{crit}$, the MSA will obtain the first tentative decision before decoding is terminated.

This property is especially important because (1) it guarantees that the MSA achieves erasurefree decoding, and (2) it points out a practical relationship between the computational limit and stack storage. Consequently, it helps in determining C_{lim} so the MSA achieves erasurefree decoding at the lowest cost of storage and computational effort.

Property 5: The probability P(u>v) that the number of stacks needed to reach the first tentative decision exceeds some number v decreases exponentially with v for sufficiently large value of v. The probability $P(C>C_v)$ that the number of computations C needed to reach the first tentative decision is greater than some number C_v decreases exponentially with C_v if C_v is the number of computations executed before stack v overflows:

$$C_v = Z_1 - 1 + \sum_{i=2}^{v} (Z_i - T) \tag{21}$$

Proof: First we calculate the number of computations (node extensions) executed at the moment of overflow. At the first stack, exactly Z_1-1 computations are performed before it overflows. At the moment of overflow, the top T entries are transferred to the second stack where decoding continues. Since these T entries occupy the second stack, only Z_2-T node extensions are permitted within the second stack before it overflows. Similarly, only Z_i-T node extensions can be performed at the i^{th} stack. The total C_v executed before stack v overflows is accumulated as equation (21).

The proof of the exponential nature of P(u>v) is given by Chevillat in reference [14]. Consequently, the number of computations before overflow of the v^{th} stack as given in equation (21) can be applied to show that the number of computations is also exponentially distributed, for a first tentative decision, or equivalently erasurefree decoding.

Most of the properties of the MSA have been discussed above for T=1, but they can be generalized for T>1. Although proofs have not been developed for T>1, we have noticed from simulation that the selection of 1<T<4 has very little effect on the error performance.

SECTION VI

SIMULATED PERFORMANCE OF THE MSA

In comparison with the Viterbi algorithm, the MSA also achieves erasurefree convolutional decoding, but with a modest decoding effort, at the expense of increased memory requirements. But the steadily decreasing cost and increasing capacity of microcomputer memories makes this tradeoff appear worthwhile. In contrast, the Viterbi maximum likelihood algorithm has a computational effort fixed by the constraint length. This computational effort grows exponentially as the constraint length is increased. Increased memory cannot be employed as efficiently by the Viterbi algorithm as by the MSA. In this section, we present simulation results showing the performance of the MSA relative to the Viterbi algorithm. Basically, we have concluded that similar error rates can be maintained at faster throughput with similar computational complexity by using the MSA.

6.1 Parameter Values

The residual error performance was measured during all trials as a function of a binary symmetric channel input signal-to-noise ratio (SNR) in the range of 4.5 to 7 dB. Equivalently, the conditions shown in Table V apply for a gaussian channel crossover probability in the range of $0.013 \leq p \leq 0.045$.

The parameters (T, Z_1, Z_i, C_{lim}) of the MSA were determined in accordance with the discussion of Section V. The values of Z_1 and C_{lim} were selected primarily to achieve optimum performance with available memory in the Z-80. In addition, Z_1 was chosen to fit the Z-80 memory and C_{lim} was made compatible with (or larger than) C_{crit}

TABLE V

General Parameter Values for (2,1)8 MSA and (2,1)15 MSA Simulations

	Algorithms / Parameters	(2,1)8 MSA	(2,1)15 MSA
CODE	free distance, d_{free}	10	17
	generator polynomial, G_1	$(1344)_8$	$(141512)_8$
	generator polynomial, G_2	$(1434)_8$	$(136266)_8$
DATA	total number of information bits processed	$> 10^6$	$> 10^6$
	number of information bits per frame, k	64	64
	number of code bits per frame, n_d	142	156
MSA	first stack size, Z_1	1024	2900
	computational limit, C_{lim}	1700	3600
	secondary stack size, Z_i	11	11
	number of nodes, transferred, T	3	3

to obtain erasurefree decoding. The discussion below concerns four parameters which determine the overall performance of the MSA: (1) residual probability of error, (2) required stack size or storage and (3) average number of decoding steps or computations, or equivalently, (4) throughput. We shall see later that (3) and (4) are interchangeable as measures of decoder speed.

6.2 Error Performance of the MSA

Sequential decoding has the characteristic that when used with appropriate tree codes to signal over memoryless channels the probability of error decreases exponentially toward zero at all transmission rates less than channel capacity. For the MSA, which is a new type of sequential decoding, we expect this behavior.

Before obtaining experimental results that typically represent the MSA performance, various simulations were performed to select adequate decoding parameters, especially those which influence the error rate. We found that although P_E decreases with increasing Z_1, and also decreases with increasing C_{1im}, Z_1 and C_{1im} play completely different roles in the MSA decoding. Z_1, the first stack size, should be made large enough to approximate SSA decoding for most blocks so that few blocks would necessitate secondary stack operations. For those very noisy blocks, C_{1im} is made at least equal to C_{crit} above which erasurefree decoding is achieved. Therefore, C_{1im} allows at least one tentative decision during secondary stack processing before the machine halts because of saturation. We found that this aspect of the MSA improves the error performance (by about 20%, on the average during all trials for practical error rates), in comparison with a random-guess upon SSA saturation.

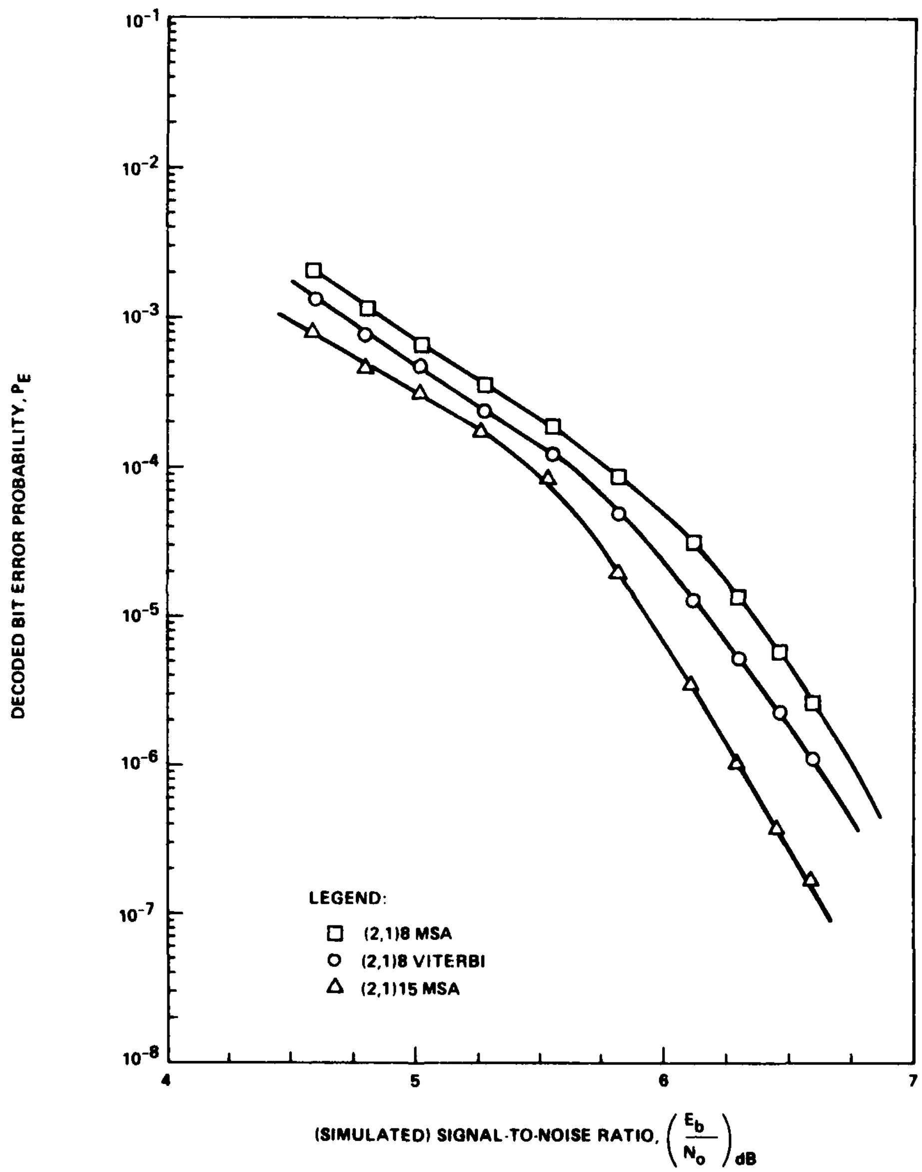
We found that it was not feasible to improve performance by linearly increasing C_{1im} any further. We believe that for larger

computational limits, the MSA will try to return to previous stacks to improve the first tentative decision. Under very noisy conditions which require secondary stacks, the correct path has such a low metric that it is buried deeply in one of the stacks. The return-to-stack operation would only tend to form more stacks without making any real contribution toward locating the correct path in the tree. Finally, the MSA reaches the larger C_{lim} and usually outputs the first tentative decision. We tested this hypothesis and found that for the (2,1)8 MSA, a value of 1700 for C_{lim} produced the same results as did a value of 5120, and that for the (2,1)15 MSA, a value of 3600 for C_{lim} gave the same results as a value of 8192. Consequently, we selected the lower C_{lim} value to achieve erasurefree decoding.

The results of the simulation for the stated parameter values are given in Tables VI through X. Curves of decoded error rate are plotted in Figure 10 in comparison with the performance of the (2,1)8 Viterbi algorithm decoder under the same channel noise environment. We concluded that the (2,1)8 Viterbi algorithm performs worse than the (2,1)15 MSA, but it clearly outperforms the (2,1)8 MSA in terms of error rate. The increase of constraint length from 8 to 15 adds modest complexity to the MSA. Both the (2,1)8 and the (2,1)15 MSA provide erasureless decoding as does the Viterbi algorithm. The MSA has additional flexibilities available. The user can adjust the parameters to a certain extent to fit the computational assets while achieving acceptable decoder error performance, and the error performance becomes asymptotically optimum if enough decoding memory is provided. Further consideration will be given to storage and throughput in the following paragraphs.

6.3 Storage Needed for the MSA

The storage requirements of the MSA relate directly to the



18-56,521

Figure 10. ERROR PERFORMANCE COMPARISONS OF (2,1) 8 MSA, (2,1) 8 VITERBI AND (2,1) 15 MSA ALGORITHMS

TABLE VI

Error Rate and Throughput Performance of the (2,1)8 MSA

Performance Parameters / $(E_b/N_o)_{dB}$	Decoded Error Rate	Throughput (BPS)
4.609	2.184×10^{-3}	160
4.812	1.155×10^{-3}	266
5.032	6.45×10^{-4}	379
5.272	3.79×10^{-4}	505
5.529	1.92×10^{-4}	630
5.823	9×10^{-5}	750
6.129	3×10^{-5}	950
6.298	1.3×10^{-5}	1050
6.465	6×10^{-6}	1126
6.669	2.8×10^{-6}	1150

TABLE VII

Error Rate and Throughput Performance of the (2,1)15 MSA

Performance Parameters / $(E_b/N_o)_{dB}$	Decoded Error Rate	Throughput (BPS)
4.609	8.16×10^{-4}	63
4.812	4.48×10^{-4}	108
5.032	3.34×10^{-4}	158
5.272	2.12×10^{-4}	233
5.529	9.1×10^{-5}	330
5.823	2.2×10^{-5}	530
6.129	3.5×10^{-6}	770
6.298	1.2×10^{-6}	900
6.465	4×10^{-7}	980
6.669	1.8×10^{-7}	1000

TABLE VIII

Error Rate and Throughput Performance of the (2,1)8 Viterbi Algorithm

Performance Parameters / $(E_b/N_o)_{dB}$	Decoded Error Rate	Throughput (BPS)
4.609	1.4×10^{-3}	50
4.812	7.72×10^{-4}	50
5.032	5×10^{-4}	50
5.272	2.3×10^{-4}	50
5.529	1.3×10^{-4}	50
5.823	5×10^{-5}	50
6.129	1.4×10^{-5}	50
6.298	6×10^{-6}	50
6.465	2.3×10^{-6}	50
6.669	1.1×10^{-6}	50

TABLE IX

Performance Comparison Under Quiet (p = 0.029) Situation

Algorithms / Performance Parameters	(2,1)8 MSA	(2,1)8 VA	(2,1)15 MSA
Decoded Error Rate	1.92×10^{-4}	1.3×10^{-4}	9.1×10^{-5}
Throughput	630 BPS	50 BPS	330 BPS
Storage	18 kBytes	2 kBytes	44 kBytes

TABLE X

Performance Comparison Under Noisy (p = 0.045) Situation

Algorithms / Performance Parameters	(2,1)8 MSA	(2,1)8 VA	(2,1)15 MSA
Decoded Error Rate	2.18×10^{-3}	1.4×10^{-3}	8.16×10^{-4}
Throughput	160 BPS	50 BPS	63 BPS
Storage	18 kBytes	2 kBytes	45 kBytes

implementation cost of the decoder. These requirements increase linearly with the constraint length while those of the Viterbi algorithm increase exponentially as shown in Figure 11. As we mentioned before, MSA memory is traded off for tolerable error performance and erasurefree decoding. We found, for example, that the available Z-80 memory (44k bytes) obtained nearly optimum performance for the (2,1)8 code. Additional memory could be used to slight advantage. For the (2,1)15 code, however, the Z-80 memory proved inadequate; the longer code required additional memory which was conveniently obtained by borrowing from a collocated NOVA minicomputer.

There are 15 bytes for each stack entry. For the (2,1)8 MSA with C_{lim} = 1700, approximately 25k bytes of memory are required for processing. For the (2,1)15 MSA, the memory requirement for C_{lim} = 3600 is 54k bytes which is quite achievable on a special purpose micro-computer system (although not available on our Z-80 which had 44k bytes available for processing after accommodating program storage). The goal of using low-cost microprocessors to implement high-performance erasureless sequential decoding seems quite attainable.

It is also helpful to talk about storage in terms of stacks. The number of stacks required for each frame (or block) is dependent upon the size of the first stack, the size of the higher-order stacks, and the computational limit. The larger the first stack is made, the less probable it is that the decoder will create secondary stacks. Thus, it is more likely that the MSA will behave like the SSA. If the higher-order stacks are made larger while fixing other parameters, the required number of stacks will be relatively smaller. However, more computations will be needed due to the increasing size of the secondary

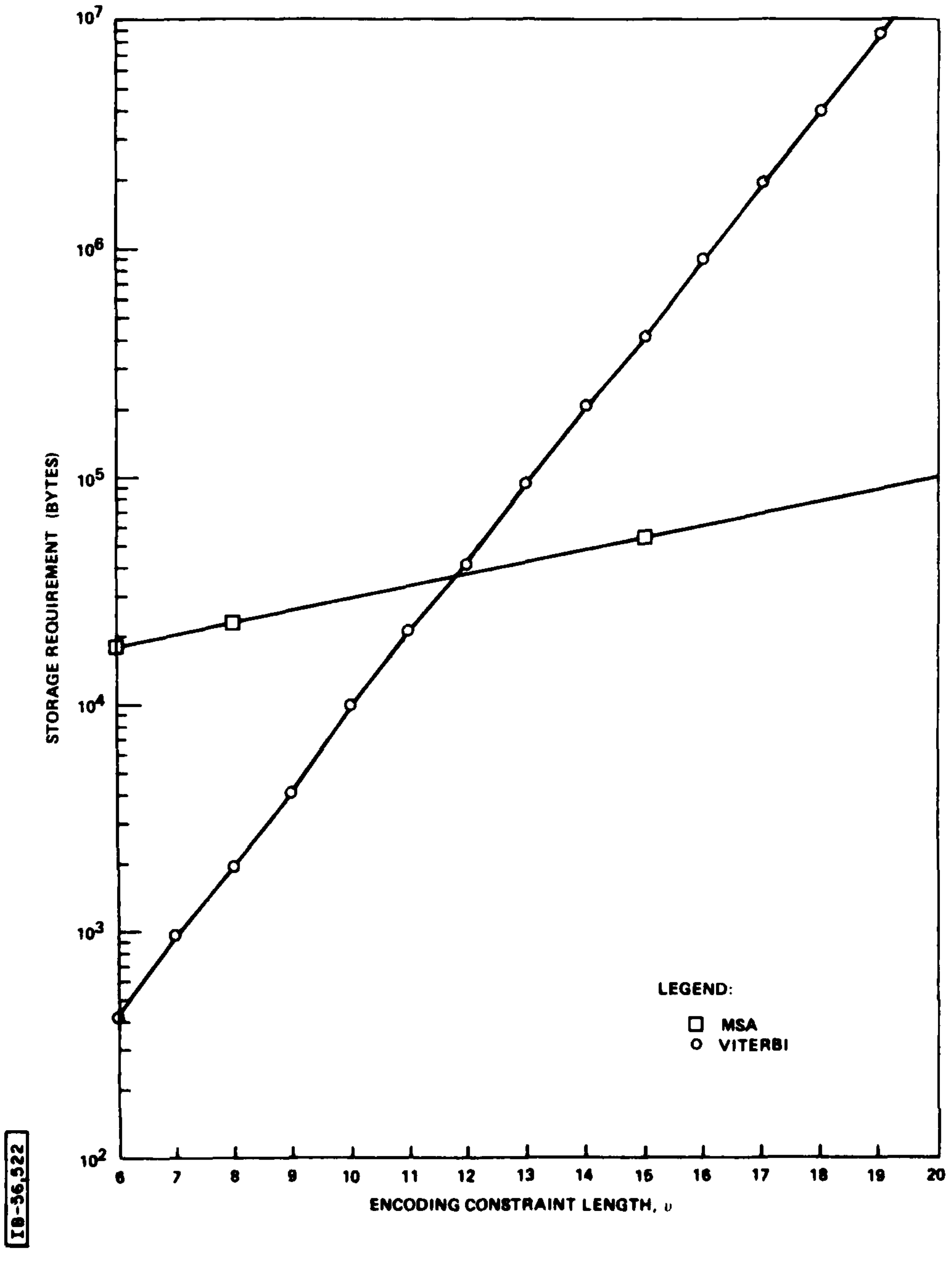
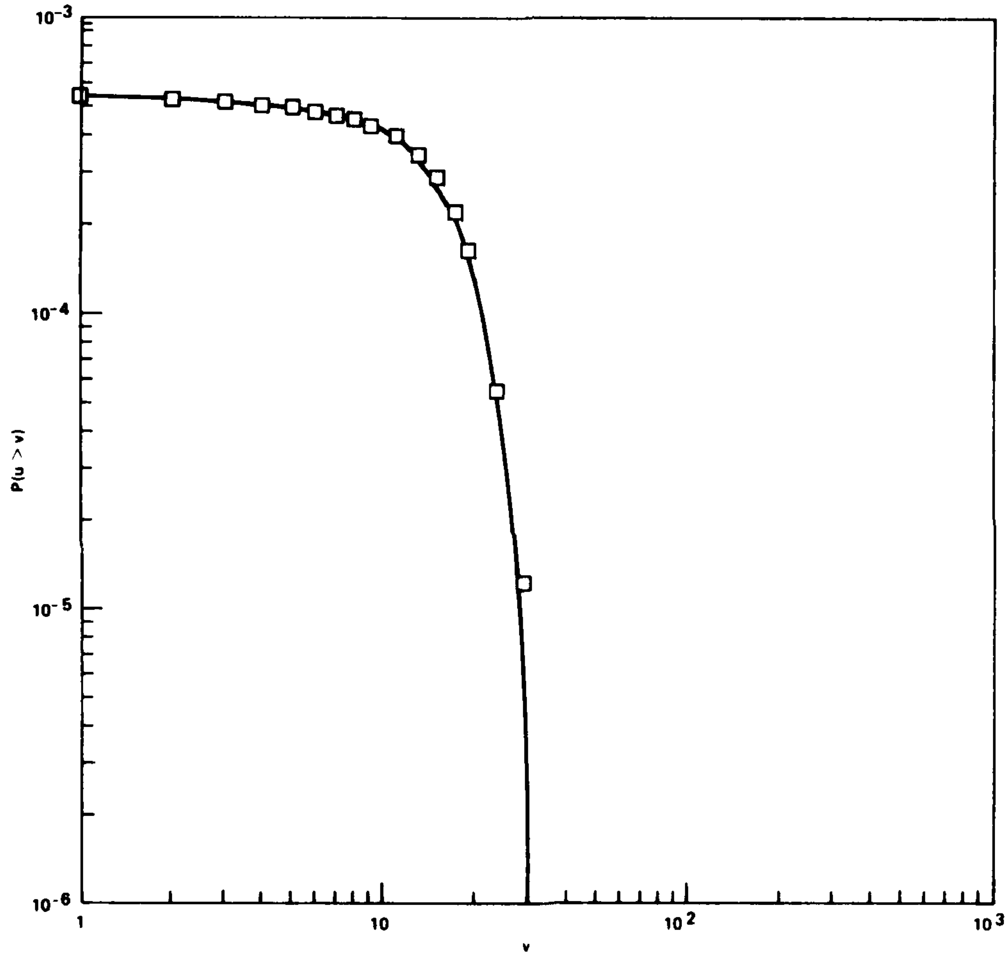


Figure 11. STORAGE REQUIREMENTS OF THE MSA AND VITERBI ALGORITHMS

stacks. The increase of computational limit will cause more stacks to be formed before the decoder terminates, which means large storage is required. For our goal of low cost and modest memory requirements, the first stack size is large while the higher-order stack size and computational limit are small (yet computational limit must exceed C_{crit} to achieve erasurefree decoding). For our selected parameters, the probability P(u>v) (that the number of stacks u needed to reach the first tentative decision exceeds some number v) is an exponentially decreasing function of v, as can be seen in Figure 12 and 13. Most of the blocks are processed with a relatively small amount of storage. Also note that the probability of SSA operation is much higher (99.95% of the time for (2,1)8 and 99.99% of the time for (2,1)15) than is the probability of forming higher-order stacks because Z_1 is large.

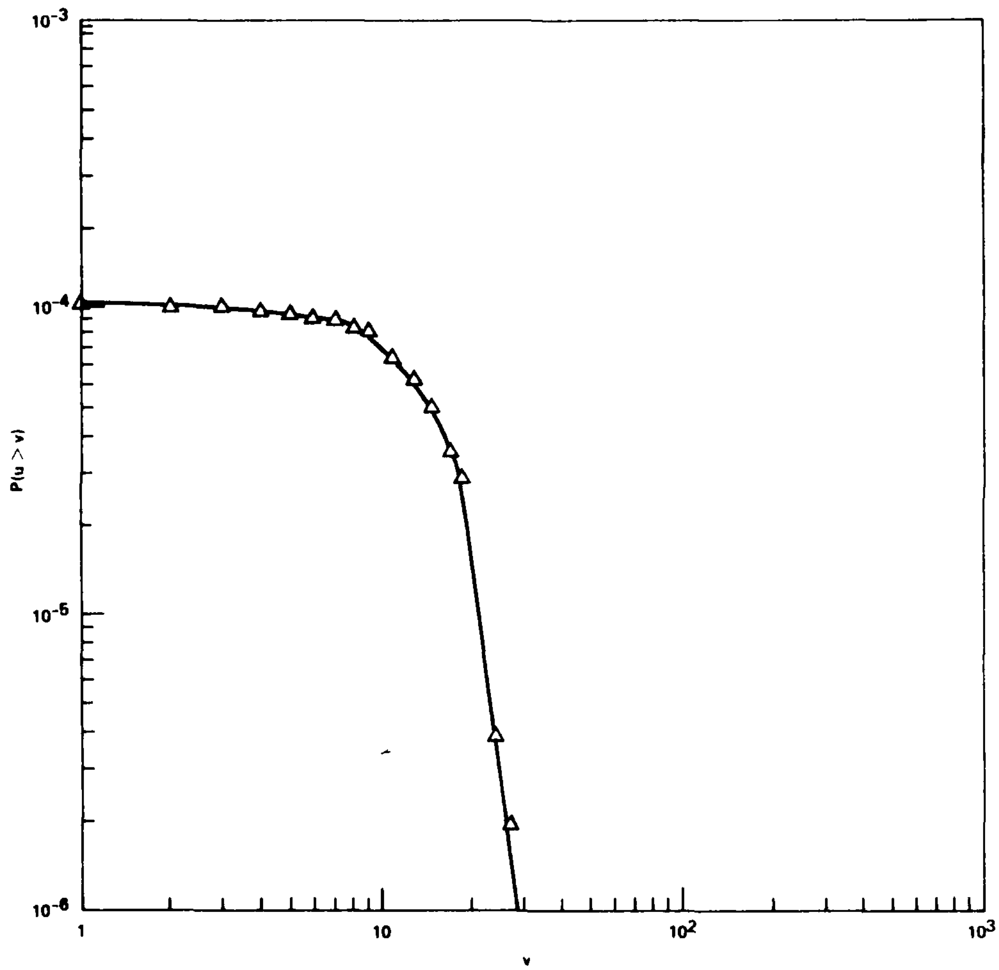
The MSA will form a number of higher-order stacks when p is large ($p = 0.045$, for example) since Z_1 is too small to approximate the SSA decoder. Once a second stack is formed, it is likely to also form other higher-order stacks. The reason is simply that the MSA is processing some badly corrupted sequences and the size of higher-order stacks is too small ($Z_i = 11$) to carry out the decoding without creating additional stacks. Z_i cannot be made large because in that case although less stacks will be required, computation effort will be too large to handle. It is fair to say that we would rather stay at the first stack all the time to finish decoding, but if the noise is severe enough to use secondary stacks, a large number of higher-order stacks should be available to process these worst-case sequences. Therefore, we have made available a number of secondary stacks in the range $0 \leq u \leq 256$.

We conclude this discussion on storage requirements with a comparison between the storage required for the MSA and that of the Viterbi algorithm. The storage required for the MSA at $p = 0.029$ is



IB-56,523

Figure 12. THE PROBABILITY DISTRIBUTION THAT THE NUMBER OF STACKS REQUIRED FOR ERASUREFREE DECODING u IS LARGER THAN SOME NUMBER v FOR (2,1)8 MSA



IB-56,524

Figure 13. THE PROBABILITY DISTRIBUTION THAT THE NUMBER OF STACKS REQUIRED FOR ERASUREFREE DECODING u IS LARGER THAN SOME NUMBER v FOR (2,1)15 MSA

about 20k bytes for the (2,1)8 code (there are 1354 entries each with a 15 byte array) and about 54k for the (2,1)15 code (there are 3153 entries each with a 15 byte array). For the (2,1)8 Viterbi algorithm, only 2k bytes are needed for storage in the Z-80 but the requirement would increase to 368k bytes if the (2,1)15 decoder were to be implemented. That is the main reason why the Viterbi algorithm is constrained to relatively short codes while the MSA can accommodate longer codes.

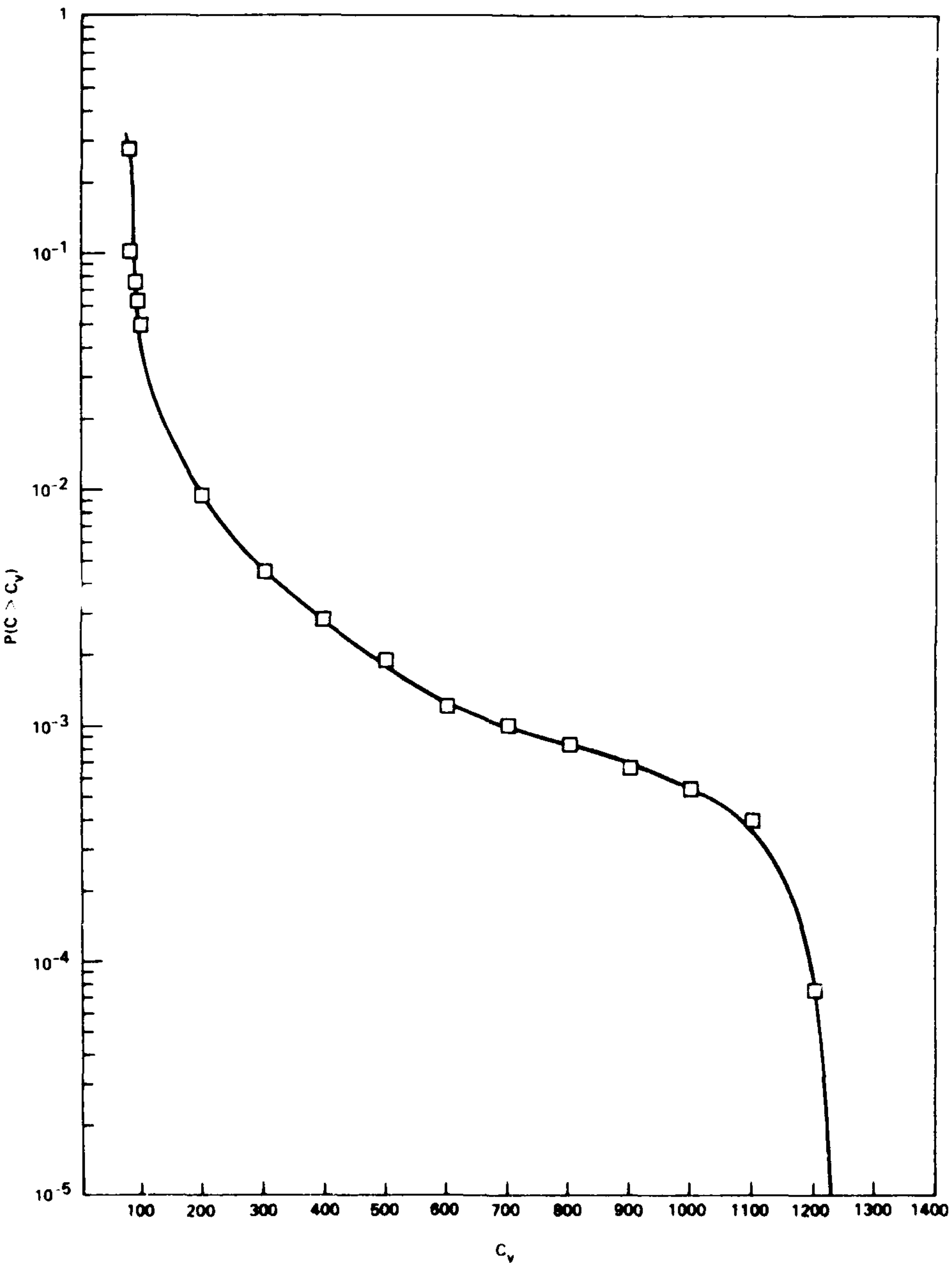
6.4 Computations and Throughput of the MSA

There are two measures of decoding speed which are closely related to each other; one is the number of computations or node extensions; the other is the actual throughput. The latter has more significance when we are comparing different decoding algorithms.

The time consumed by the decoder is approximately proportional to the number of decoder computations that are performed, where a single decoder computation of the MSA comprises all the operations performed for each node extension (including metric calculations, ordering and possibly stack transfer/deletion). For the stack algorithm the average number of decoder computations per decoder information bit is bounded by a constant for all rates less than R_{comp}. The R_{comp} is a function of the channel transition probabilities only, and exceeds one-half the channel capacity for all nonpathological memoryless channels. The average number of computations increases exponentially with k if $R > R_{comp}$. Therefore, for large k, one would ordinarily not attempt to use a stack algorithm at rates that exceed R_{comp}; i.e., for BSC crossover probability $p > 0.045$. It was stated previously in Section II that even the SSA shows a significant speed advantage over the Fano algorithm for the BSC. But the computational distribution is Pareto for the SSA.

Below R_{comp}, the search increases only linearly and it is intuitively satisfying to see that as long as the decoding work has the same character as that undertaken by the maximum likelihood decoder, the error results are the same. At rates below R_{comp}, the error performance is close to optimal, whereas the decoding effort increase is only linear with k. As the channel rate R approaches close to R_{comp} from below, the computational load increases tremendously. For $R < R_{comp}$, $P(C \geq C_v) = A C_v^{-1}$ where A is a constant of proportionality; but the average decoding computational effort becomes infinite when $R = R_{comp}$. While this does not necessarily happen in practice (the theoretical result is only a bound), the dramatic increase, even as R exceeds about 0.9 R_{comp}, has certainly been observed. For the MSA, $p\ (C > C_v)$ is an exponentially distributed function of C_v (Property 5 of Section V) as shown in Figures 14-16. The number of computations on the average is very small and for C_v considerably less than C_{crit}, $p\ (C > C_{lim})$ can be made small also. This also means the computations needed to reach the first tentative decision are almost always less than C_{lim} and erasurefree decoding is achieved.

The number of computations for the MSA is a random variable; we found the average is 1.37 computations per information bit for the (2,1)8 code and 1.41 for the (2,1)15 code. The number of computations of the Viterbi algorithm is a constant decoding effort of 128 computations per information bit for the (2,1)8 code. Although the average number of computations of the MSA is much less than that of the Viterbi algorithm (by two orders of magnitude), the operations needed to be performed for each computation (stack order/deletion/creation) are more complicated and consume more computer time than those of the simple add/compare/select operation of the Viterbi algorithm. With the software implementation of the MSA, erasurefree decoding is also obtained at the expense of some computer time (computational limit needs to be set sufficiently large). The stack



18-56,525

Figure 14. THE PROBABILITY DISTRIBUTION THAT THE NUMBER OF COMPUTATIONS PERFORMED FOR ERASUREFREE DECODING C IS LARGER THAN SOME NUMBER C_v ($64 < C_v < 1400$) FOR (2,1)8 MSA

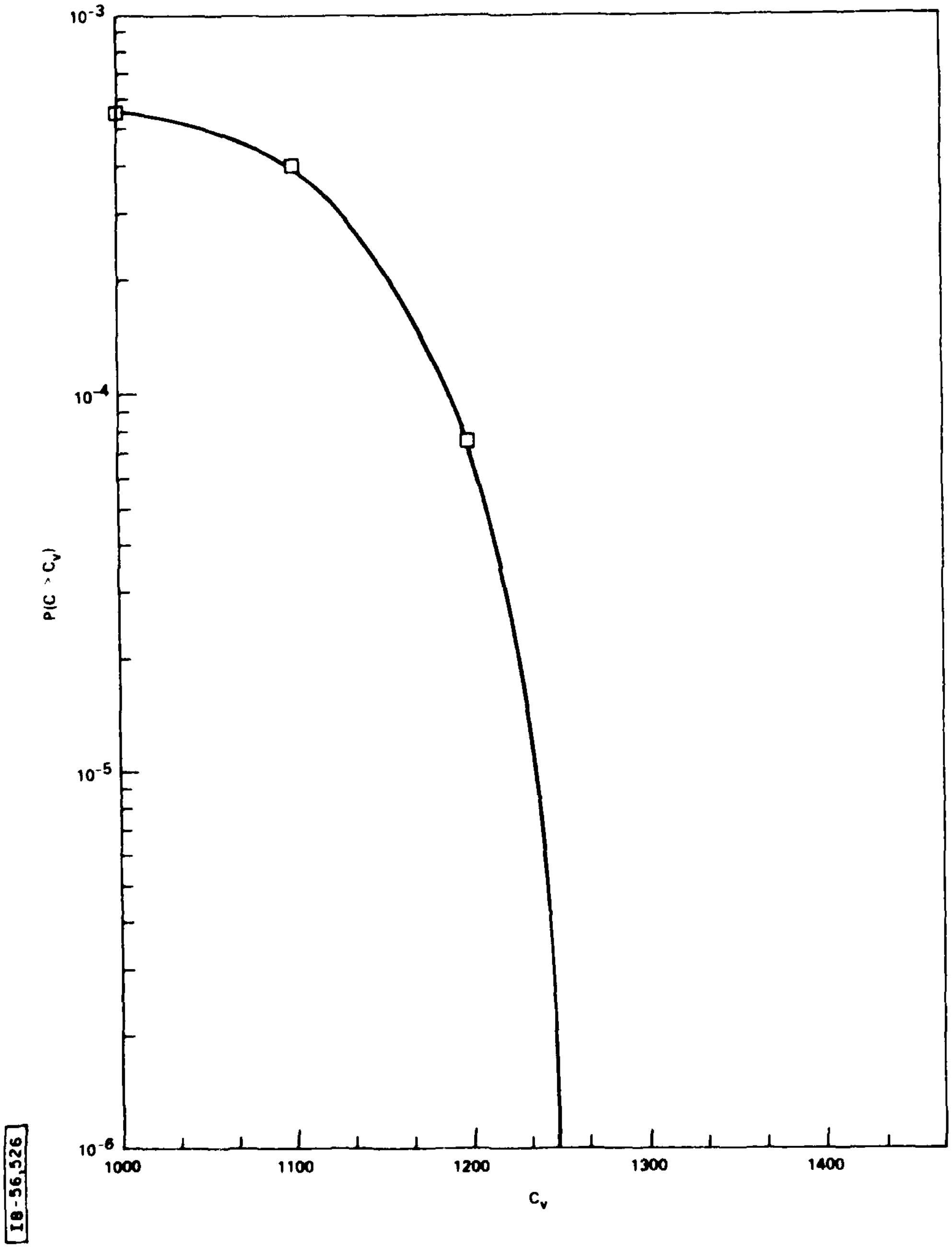


Figure 15. THE PROBABILITY DISTRIBUTION THAT THE NUMBER OF COMPUTATIONS PERFORMED FOR ERASUREFREE DECODING C IS LARGER THAN SOME NUMBER C_v ($1000 < C_v < 1400$) FOR (2, 1) 8 MSA

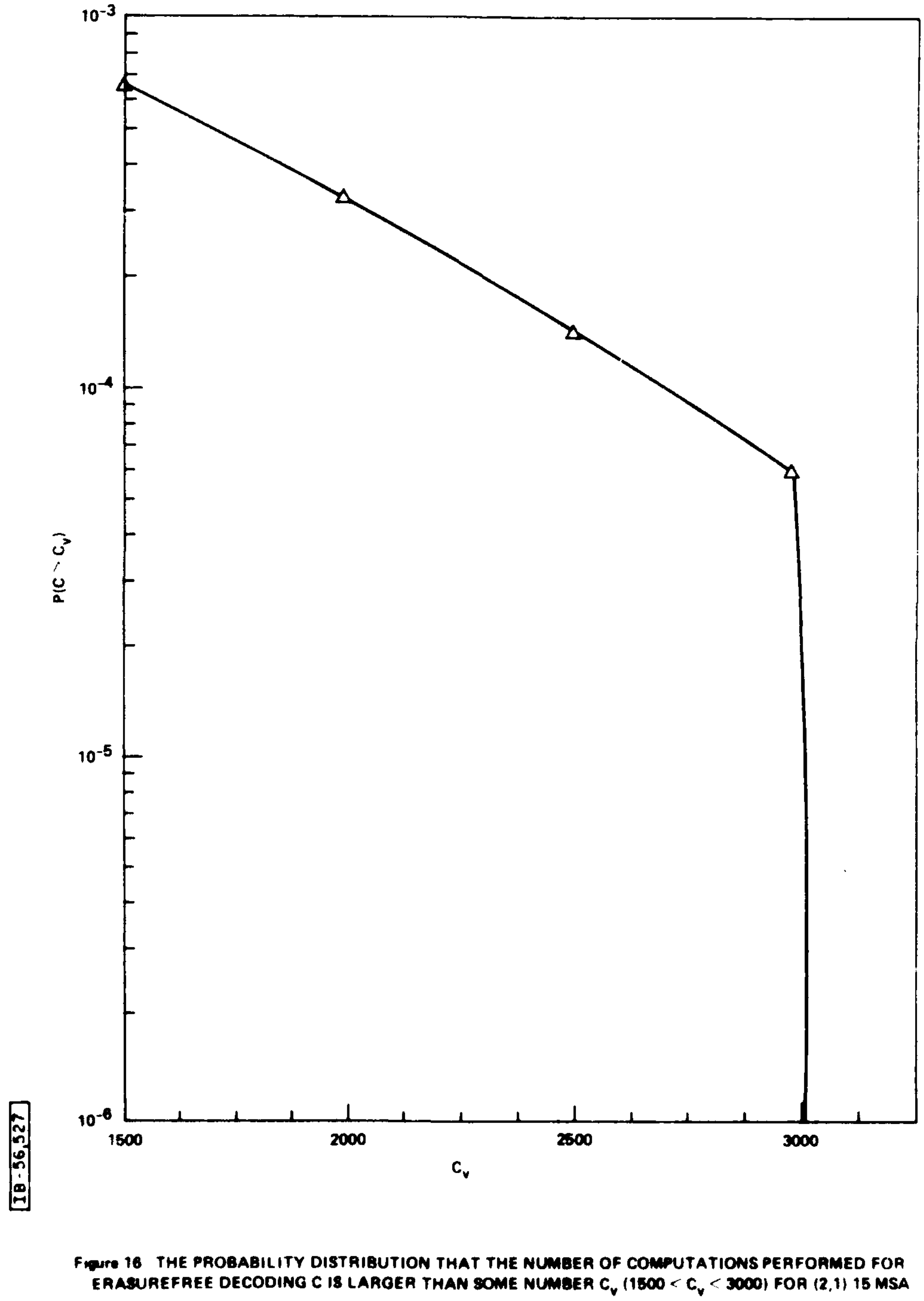


Figure 16 THE PROBABILITY DISTRIBUTION THAT THE NUMBER OF COMPUTATIONS PERFORMED FOR ERASUREFREE DECODING C IS LARGER THAN SOME NUMBER C_v ($1500 < C_v < 3000$) FOR (2,1) 15 MSA

creation/ordering/deletion operation is about 10 times slower than the add/compare/select operation.

We see a close relationship between computational limit C_{lim} and the number of stacks u. C_{lim} is actually the number of entries which must be put in u stacks. Since the first stack size is Z_1 and each secondary stack size is Z_i, the number of stacks required is therefore:

$$u = \frac{C_{lim} - Z_1}{Z_i} + 1 \qquad (22)$$

where $\frac{C_{lim} - Z_1}{Z_i}$ accounts for the number of secondary stacks and the added 1 accounts for the first stack.

If we fix the computational limit, C_{lim} (C_{lim} = 1700), the first stack size, Z_1 (Z_1 = 1024) and the secondary stack size, Z_i (Z_i = 11), the maximum number of stacks required is determined to be 63.

The second measure which gives the actual decoding speed is the throughput of the decoder. It counts, in terms of bits/second, the speed of the MSA operation while excluding the processing time of the other peripheral devices (encoder, noise generators, etc.). The throughput is a more realistic measure than the number of computations and reflects the actual speed advantage of the MSA over the Viterbi algorithm. The throughput of the (2, 1)8 MSA under moderate noise environment ($p = 0.029$) is 500 bits/second, as compared to 50 bits/second for the (2,1)8 Viterbi algorithm. Therefore, the MSA is actually about ten times faster than the Viterbi algorithm to achieve comparable error performance. A comparison of throughput is shown in Figure 17 and Table IX and X. Notice that MSA shows a varying throughput while the Viterbi algorithm gives constant throughput for various noisy conditions. These results further demonstrate the variable nature of the computational effort of the MSA.

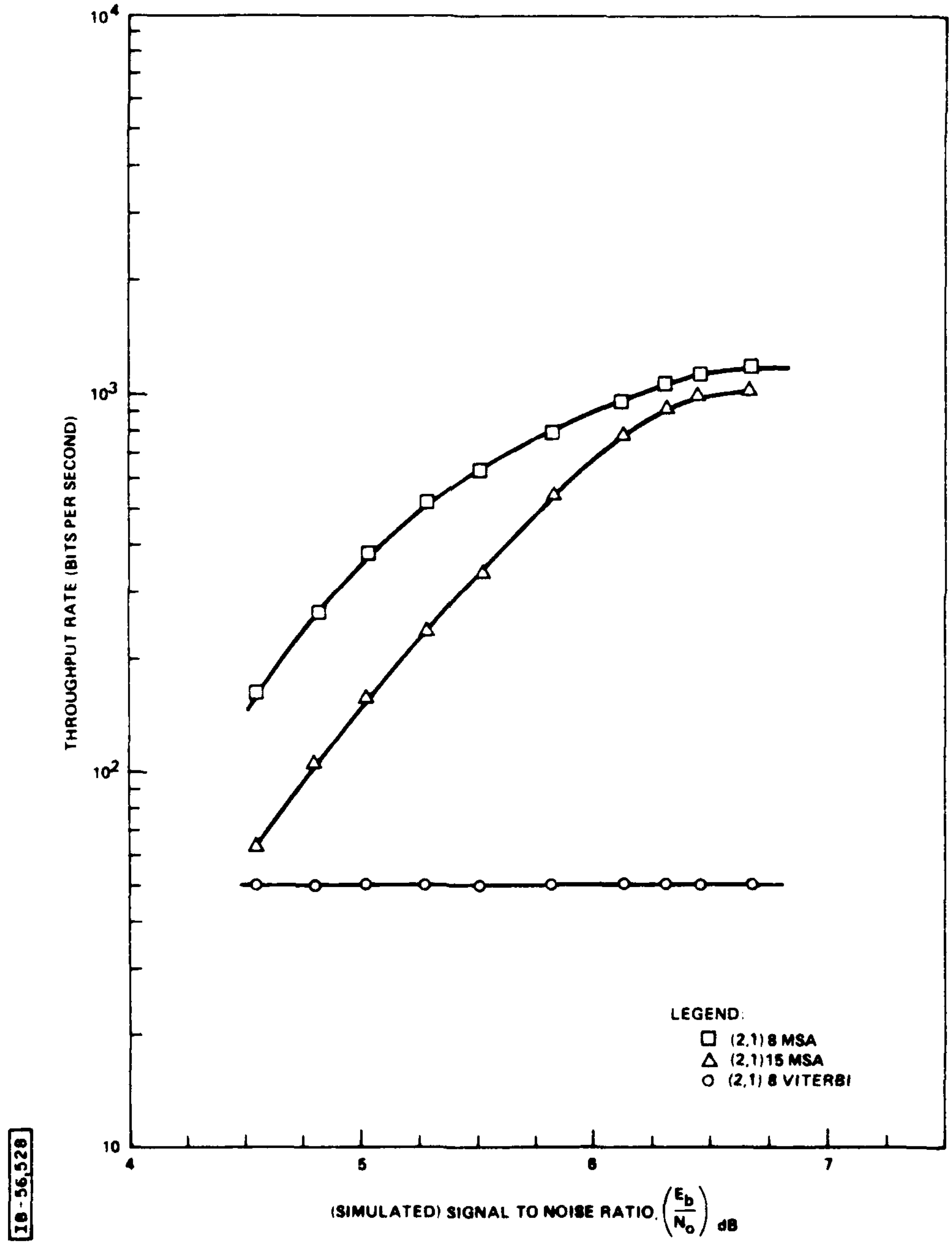


Figure 17. THROUGHPUT COMPARISONS OF (2,1)8 MSA, (2,1) 15 MSA AND (2,1)8 VITERBI ALGORITHMS

SECTION VII

CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK

In this report, we have examined a software implementation of the Multiple Stack Sequential Decoding Algorithm (MSA) on the Z-80 microcomputer system and have discussed the resulting performance subject to certain rules of parameter selection. Techniques have been described for maintaining a tolerable undetected error probability as the SNR decreases. The utilization of the MSA has eliminated the erasure problem caused by incomplete decoding in other sequential decoding procedures. The major limiting property of the Single Stack Algorithm (SSA), Pareto computational distribution, has been removed by processing multiple stacks arranged in parallel; the computational distribution is now an exponentially decreasing function of some number C_v. The feasibility of implementing both the Viterbi algorithm and the MSA on the Z-80 system has also been verified. In this section, we will draw some conclusions and make suggestions based on the following:

1. performance comparison of MSA and Viterbi algorithm with respect to:
    - a. decoded error rate,
    - b. decoding effort,
    - c. storage requirement,
2. soft quantization possibilities for MSA,
3. real-time (2,1)8 MSA decoder possibilities and feasibility requirements,
4. performance of the MSA on a burst noise model,
5. suggested further research areas.

7.1 Performance Comparisons of the MSA and Viterbi Algorithm

The performances of the MSA and the Viterbi algorithm have been compared in respect to the decoded error rate, the decoding effort (or throughput rate) and the storage requirements. We have concluded that the decoded error rate of the (2,1)8 MSA is slightly worse (less than 0.5dB) than that of the (2,1)8 Viterbi algorithm, but the throughput rate is ten times faster when implemented on the Z-80. If the user can tolerate some degradation of error performance to gain more speed, the (2,1)8 MSA, with a storage requirement ten times larger than that of Viterbi algorithm, is certainly an effective alternative. The (2,1)15 MSA performs better than the (2,1)8 Viterbi algorithm with respect to decoded errors, while the average throughput rate is still somewhat faster. However, the storage requirement is about twenty times larger. But, considering the declining cost of microcomputer memory, the (2,1)15 MSA may constitute the most effective alternative to a (2,1)15 Viterbi algorithm if a complete decoding method, which achieves low error probabilities at acceptable speeds, is desired.

7.2 Soft Quantization Possibilities for MSA

For the Viterbi algorithm, soft quantization is easy to implement and is a good practice. The 2 dB gain is usually obtained by processing the soft quantized data at the input section, and then is incorporated in the branch and path metrics without requiring additional storage. The MSA, on the other hand, must store several thousand branches of data according to the size of the stack; each would contain $\log_2 \frac{Q}{R_c}$ bits for rate R_c and Q-level quantization. The complicated operations of the decoder, such as stack creation, stack transfer, and stack deletion, make the task of soft quantization even more unpredictable. The 2 dB improvement promised by theory might not be worth the added complexity.

7.3 Real-Time (2,1)8 MSA Feasibility Requirements

The MSA was tested using mainly software implementations with a simulated channel model. If a special-purpose hardware decoder and peripheral devices were used, the data rate could have been high enough for real-time consideration and on-line processing low data-rate applications where special-purpose Viterbi decoders are currently being used. As for the software Z-80 MSA decoder, only quite low data rate was possible because of the speed limit of the Z-80 (in the range of a few hundred bits per second). The software Viterbi algorithm achieves data rates of only a few tens of bits per second on the same machine. The combination of the MSA and next-generation processors may prove to be more rewarding.

Judging from the decoding operations of the MSA, it would be more timesaving for the microcomputer system to have the following instructions.

(1) Compare

(2) Decrement and jump (an instruction that decrements an index register and jumps if the register content is non-zero).

The presence of more than one accumulator would reduce the decoding time even more. The choice of the Z-80 type of system is justified in this aspect. A few other features that would be helpful to achieve real-time decoding are:

(1) all computer time devoted to MSA decoding,

(2) maximum computations, C_{lim}, varied according to allowable processing time,

(3) adjustable delay for information delivery (i.e., variable buffer),

(4) faster instruction cycle time with a large speed advantage to keep up with incoming data,

(5) different machines operating under a central control, each performing a single operation such as stack ordering, stack deletion, or stack transfer, or

(6) different machines operating independently, each performing several operations and each exploring different paths in the code tree.

A 16-bit microcomputer is suitable for feature 4. The bit-slice bipolar microcomputer is appropriate for features 5 or 6.

Because it is the message that cannot be decoded that limits the real-time decoder performance, it is also this message which must be considered when speed and memory requirements are determined for the MSA. All codewords could be successfully and completely decoded given enough memory and processing time for rates less than channel capacity. Even with a somewhat constrained computational limit, the MSA is able to obtain some decision which is better than the random-guess version of the finite-stack SSA.

7.4 Performance of the MSA on Burst Noise Model

It was expected that sequential decoding would be unsuitable as a burst-correcting technique. The variability of the decoding computation time prohibits its successful utilization as a burst decoder. A burst model was implemented to further verify that this consideration applies also to the MSA. The error rate and throughput performance showed that the MSA cannot handle bursts at all and interleaving and de-interleaving techniques as are required with the Viterbi algorithm must be applied to obtain tolerable performance.

7.5 Suggestions for Further Study

Further study in the use of the multiple stack algorithm as a low-cost, efficient sequential decoding method should include the design of a real-time MSA decoder at useful data rates. It should also consider the implementation of the MSA on either a faster 16-bit NMOS microprocessor (the Z-8000, for instance) or on a bit-slice bipolar microprocessor suitable for parallel processing.

LIST OF REFERENCES

1. Chevillat, P. R., and Costello, D. J., "A Multiple Stack Algorithm for Erasurefree Decoding of Convolutional Codes," IEEE Transaction on Communications, Vol. COM-25, pp. 1460-1470, December 1977.

2. Jacobs, I. M. and Berlekamp, E. R., "A Lower Bound to the Distribution of Computation for Sequential Decoding," IEEE Transaction on Information Theory, Vol. IT-13, pp. 167-174, April 1967.

3. Massey, J. L., "Variable-Length Codes and the Fano Metric", IEEE Transaction on Information Theory, Vol. IT-18, pp. 196-198, January 1972.

4. Fano, R. M., "A Heuristic Discussion of Probabilistic Decoding," IEEE Transaction on Information Theory, Vol. IT-9, pp. 64-74, April 1963.

5. Jelinek, F., "A Fast Sequential Decoding Algorithm Using a Stack," IBM Journal of Research and Development, Vol. 13, pp. 675-685, November 1969.

6. Heller, J. A., and Jacobs, I. M. "Viterbi Decoding for Satellite and Space Communication," IEEE Transaction on Communications, Vol. COM-19, pp. 835-848, October 1971.

7. Viterbi, A. J., "Error Bounds for Convolutional Codes and an Asypototically Optimum Decoding Algorithm," IEEE Transaction on Information Theory, Vol. IT-13, pp. 260-269, April 1967.

8. Forney, G. D., Jr., "Concatenated Codes," MIT Press Research Monograph 37, 1966.

9. Chevillat, P. R., and Costello, D. J., "An Analysis of Sequential Decoding for Specific Time - Invariant Convolutional Codes," IEEE Transaction on Information Theory, Vol. IT-24, pp. 443-451, July 1978.

10. Wozencraft, J. M., and Reiffen, B., Sequential Decoding, The MIT Press, Cambridge, MA 1961.

11. Massey, J. L., and Costello, D. J., "Nonsystematic Convolutional Codes for Sequential Decoding in Space Applications," IEEE Transaction on Communication Technology, Vol. COM-19, pp. 806-813, October 1971.

12. Carhoun, D. O., "Generalized Cyclic Codes: Volume I, Overview and Technical Summary," The MITRE Corporation, MTR-3227, Vol. I, October 1977.

13. Bahl, L. R., and Jelinek, F., "Rate 1/2 Convolutional Codes with Complementary Generators," IEEE Transaction on Information Theory, Vol. IT-17, pp. 718-727, November 1971.

14. Chevillat, P. R., "Fast Sequential Decoding and a New Complete Decoding Algorithm," PhD Dissertation, Department of Electrical Engineering, Illinois Institute of Technology, May 1976.

15. Vanelli, J. C., and Shehadeh, N. M., "Computation of Bit Error Probability Using the Trapezoidal Integration Rule," IEEE Transaction on Communication Technology, Vol. COM-22, pp. 331-334, March 1974.

16. Johannesson, R., "On the Error Probability of General Trellis Codes With Applications to Sequential Decoding," IEEE Transaction on Information Theory, Vol. IT-23, pp. 609-611, September 1977.

17. Linkabit Corporation, Final Report on a Coding Systems Study for High Data Rate Telemetry Links, Contract NAS 2-6024, NASA Ames Research Center, Moffett Field, California.

18. Forney, G. D., Jr., "Convolutional Codes III: Sequential Decoding, Information and Control, Vol. 25, pp. 267-297, July 1974.

19. Wozencraft, J. M., and Jacobs, I. M., Principles of Communication Engineering, Wiley, New York, 1965, pp. 405-438.

20. Gallager, R. G., Information Theory and Reliable Communication, Wiley, New York, 1968, pp. 258-286.

21. Peterson, W. W., and Weldon, E. J., Error Correcting Codes, The MIT Press, Cambridge, MA, 1972 (Second Edition), pp. 392-426.

22. Viterbi, A. J., and Omura, J. K., Principles of Digital Communication and Coding, McGraw-Hill, New York, 1968, pp. 227-381.

23. Knuth, D. E., The Art of Computer Programming, Vol. 2 (Seminumerical Algorithm), Addison Wesley, Reading, MA, 1969, pp. 9-10.

DISTRIBUTION LIST

INTERNAL

D-10

E. L. Key

D-11

A. J. Roberts

D-53

R. K. Boatman

D-73

J. A. Clapp

D-80

R. W. Jacobus
S. J. Rabinowitz

D-82

A. Arcese
A. Bark
J. E. Blanchard
W. M. Bridge
D. R. Bungard
P. A. Buote
B. D. Campbell
D. O. Carhoun (5)
A. H. Chan
R. J. Cosentino
A. L. Fullerton
G. L. Glatfelter
R. D. Haggarty
J. R. Hamalainen
N. Houlding
S. N. Hunt
B. L. Johnson
V. Kross
I. LaGarde
H. H. Ma (15)
R. A. McCown
F. L. McDonald
T. J. McDonald

D-82 (continued)

S. J. Meehan
A. L. Murphy
E. A. Palo
C. E. Pearson
B. D. Perry
M. A. Poole
J. H. Reisert
G. O. Sauermann
L. E. Smith
W. K. Talley
G. B. Tiffany
S. M. Waldstein
S. S. Weinrich
M. R. Weiss
J. C. Williamson

D-86

C. H. Chen
R. E. Clapp
J. D. R. Kramer
R. J. Preuss
J. F. Sullivan

D-96

E. N. Skoog

EXTERNAL

ESD

Dr. D. B. Brick (XR)
Capt. P. H. Wendzikowski (XRE)

RADC-ET

Dr. J. Silverman (ESE) (5)

RADC-DC

Dr. F. D. Schmandt (DCCT)

END

DATE

FILMED

11-80

DTIC